



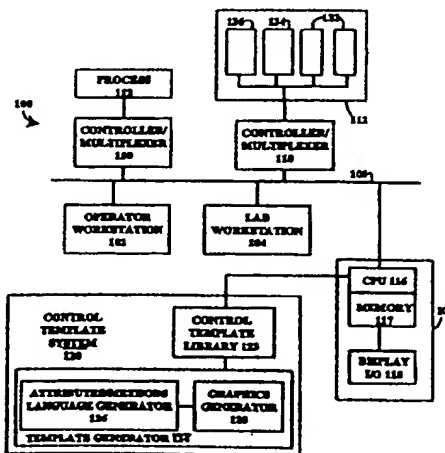
INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

| | | | |
|---|--|--|---|
| (51) International Patent Classification 6 : G05B 19/418 | | A2 | (11) International Publication Number: WO 98/36335 (43) International Publication Date: 20 August 1998 (20.08.98) |
| (21) International Application Number: PCT/US98/01573 (22) International Filing Date: 6 February 1998 (06.02.98) (30) Priority Data: 08/799,966 14 February 1997 (14.02.97) US | | John, R.; 1808 Gaylord Drive, Austin, TX 78728 (US); CHRISTENSEN, Dan, D.; 9001 Marthas Drive, Austin, TX 78717 (US). SCHLEISS, Duncan; 8115 Cardin Road, Austin, TX 78759 (US). (74) Agents: KOESTNER, Ken, J. et al.; Skjerven, Morrill, MacPherson, Franklin & Friel LLP, Suite 700, 25 Metro Drive, San Jose, CA 95110 (US). | |
| (71) Applicant: FISHER-ROSEMOUNT SYSTEMS, INC. [US/US]; 8301 Cameron Road, Austin, TX 78754 (US). | | (81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG). | |
| (72) Inventors: NIXON, Mark; 1503 Blackjack Drive, Round Rock, TX 78681 (US). HAVEKOST, Robert, B.; 14507 Crystal Court, Austin, TX 78728 (US). JUNDT, Larry, O.; 305 Northfield, Round Rock, TX 78681 (US). STEVENSON, Dennis; 16904 Sabertooth Drive, Round Rock, TX 78681 (US). OTT, Michael, G.; 10216 Talleyran Drive, Austin, TX 78750 (US). WEBB, Arthur; 7 Pytchley Drive, Loughborough, Leicestershire LE11 2RH (GB). LUCAS, Mike; 10 Derby Close, Broughton Astley, Leicestershire LE9 6NF (GB). HOFFMASTER, James; 8514 Long Canyon Drive, Austin, TX 78730 (US). OTTENBACHER, Ron; 9301 Rolling Oaks Trail, Austin, TX 78750 (US). BEOUGHTER, Ken, J.; 16917 Korat Lane, Round Rock, TX 78681 (US). FALTESEK, Roy; 2202 Falcon Drive, Round Rock, TX 78681 (US). KRIVOSHEIN, Ken, D.; 108 Elgin Woods Lane, Elgin, TX 78621 (US). SHEPARD, | | Published Without international search report and to be republished upon receipt of that report. | |

(54) Title: PROCESS CONTROL SYSTEM USING A LAYERED-HIERARCHY CONTROL STRATEGY DISTRIBUTED INTO MULTIPLE CONTROL DEVICES

(57) Abstract

A process controller (100) implements smart field device standards (132) and other bus-based architecture standards so that communications and control among devices are performed and the standard control operations are transparent to a user. The process controller implements and executes a standard set of function blocks (522) or control functions defined by a standard protocol so that standard-type control is achieved with respect to non-standard-type devices (12). The process controller enables standard devices (6) to implement the standard set of function blocks and control functions. The process controller implements an overall strategy as if all connected devices are standard devices by usage of a Fieldbus function block as a fundamental building block for control structures. Function blocks are defined to create control structures for all types of devices. A user defines the control strategy by building a plurality of function blocks and control modules (440) and downloading or installing user-specified portions of the control strategy into the Fieldbus devices and the non-Fieldbus devices. Thereafter, the Fieldbus devices automatically perform the downloaded portions of the overall strategy independently of other portions of the control strategy. The process control system includes a diagnostic monitoring and display functionality for viewing, in a coherent manner, diagnostic information relating to a process that operates over multiple devices and system components. The digital control system automatically senses when a new controller is attached to a network and determines the number and types of I/O Ports that are attached to the new controller. The digital control system program also includes an automatic configuration program that responds to sensing of a new controller by automatically configuring the input/output (I/O) subsystem. Upon connection of the device, the device is automatically sensed and configured using the database configuration information, without setting of physical switches or node address information on the devices. The digital control system with a predetermined configuration automatically senses the connection to a network of a digital device that is not included in the predetermined configuration. The process control system includes a user interface (300) which supports multiple IEC-1131 standard control languages and user-selection from among the control languages. From a single application routing, a user selects a control language from among a plurality of control languages including, for example, Function Blocks, Sequential Function Charts, Ladder Logic and Structural Text, to implement a control strategy. The process control system includes an alarm and event monitoring and display system for which various users of the system can easily prioritize the alarm and event information that is displayed.



Systems that perform, monitor, control, and feed back functions in process control environments are typically implemented by software written in high-level computer programming languages such as Basic, Fortran or C and executed on a computer or controller. These high-level languages, although effective for process control programming, are not usually used or understood by process engineers, maintenance engineers, control engineers, operators and supervisors. Higher level graphical display languages have been developed for such personnel, such as continuous function block and ladder logic. Thus each of the engineers, maintenance personnel, operators, lab personnel and the like, require a graphical view of the elements of the process control system that enables them to view the system in terms relevant to their responsibilities.

For example, a process control program might be written in Fortran and require two inputs, calculate the average of the inputs and produce an output value equal to the average of the two inputs. This program could be termed the AVERAGE function and may be invoked and referenced through a graphical display for the control engineers. A typical graphical display may consist of a rectangular block having two inputs, one output, and a label designating the block as AVERAGE. A different program may be used to create a graphical representation of this same function for an operator to view the average value. Before the system is delivered to the customer, these software programs are placed into a library of predefined user selectable features. The programs are identified by function blocks. A user may then invoke a function and select the predefined graphical representations to create different views for the operator, engineer, etc. by selecting one of a plurality of function blocks from the library for use in defining a process control solution rather than having to develop a completely new program in Fortran, for example.

A group of standardized functions, each designated by an associated function block, may be stored in a control library. A designer equipped with such a library can design process control solutions by interconnecting, on a computer display screen, various functions or elements selected with the function blocks to perform particular tasks. The microprocessor or computer associates each of the functions or elements defined by the function blocks with predefined templates stored in the library and relates each of the program functions or elements to each other according to the interconnections desired by the designer. Ideally, a designer could design an entire process control program using graphical views of predefined functions without ever writing one line of code in Fortran or other high-level programming language.

One problem associated with the use of graphical views for process control programming is that existing systems allow only the equipment manufacturer, not a user of this equipment, to create his own control functions, along with associated graphical views, or modify the predefined functions within the provided library.

New process control functions are designed primarily by companies who sell design systems and not by the end users who may have a particular need for a function that is not a part of the standard set of functions supplied by the company. The standardized functions are contained within a control library furnished with the system to the end user. The end user must either utilize existing functions supplied with

the design environment or rely on the company supplying the design environment to develop any desired particular customized function for them. If the designer is asked to modify the parameters of the engineer's view, then all other views using those parameters have to be rewritten and modified accordingly because the function program and view programs are often developed independently and are not part of an integrated development environment. Clearly, such procedure is very cumbersome, expensive, and time-consuming.

Another problem with existing process control systems is a usage of centralized control, typically employing a central controller in a network, executing a program code that is customized for specialized, user-defined control tasks. As a result, the process control systems are typically constrained to a particular size and difficult to adapt over time to arising needs. Similarly, conventional process control systems are inflexible in configuration, often requiring a complete software revision for the entire system when new devices are incorporated. Furthermore, the conventional process control systems tend to be expensive and usually perform on the functions initially identified by a user or a system designer that are only altered or reprogrammed to perform new functions by an expert who is familiar with the entire control system configuration and programming.

What is needed is a uniform or universal design environment that can easily be used, not only by a designer or manufacturer but also a user, to customize an existing solution to meet his specific needs for developing process control functions. What is further needed is a personal computer-based process control system that is easily implemented within substantially any size process and which is updated by users, without the aid of the control system designer, to perform new and different control functions.

Many process control systems include local field devices such as valves, motors, regulators and the like which are responsive to specific control protocols, such as Profibus, Fieldbus, CAN and the like, to implement various control function routines. Accordingly, these controllers are responsive to certain standard control protocols to implement control functionality in the field. The use of such standard control signal protocols can reduce the time and effort of developing a control system because a designer can use the same types of control signals from all devices responsive to the control protocol.

However, certain control devices are not responsive to standard control protocols. These devices are often responsive to other types of control signals such as digital ON/OFF signals, analog current signals or analog voltage signals. A system designer either has to avoid using field devices that are nonresponsive to an installed protocol, or develop systems that operate under one or more protocols. Thus, present day processing systems disadvantageously lack a capability to utilize both standard protocol control devices and devices that do not respond to control signals defined under the standard protocols.

What is needed is a process control system that controls both devices that are defined using a standard protocol and other, non-protocol devices in a manner that is transparent to the user of the process control system.

Systems that perform, monitor, control, and feed back functions in process control environments are typically implemented by software written in high-level computer programming languages such as Basic, Fortran or C and executed on a computer or controller. These high-level languages, although effective for process control programming, are not usually used or understood by process engineers, maintenance engineers, control engineers, operators and supervisors. Higher level graphical display languages have been developed for such personnel, such as continuous function block and ladder logic. Thus each of the engineers, maintenance personnel, operators, lab personnel and the like, require a graphical view of the elements of the process control system that enables them to view the system in terms relevant to their responsibilities.

For example, a process control program might be written in Fortran and require two inputs, calculate the average of the inputs and produce an output value equal to the average of the two inputs. This program could be termed the AVERAGE function and may be invoked and referenced through a graphical display for the control engineers. A typical graphical display may consist of a rectangular block having two inputs, one output, and a label designating the block as AVERAGE. A different program may be used to create a graphical representation of this same function for an operator to view the average value. Before the system is delivered to the customer, these software programs are placed into a library of predefined user selectable features. The programs are identified by function blocks. A user may then invoke a function and select the predefined graphical representations to create different views for the operator, engineer, etc. by selecting one of a plurality of function blocks from the library for use in defining a process control solution rather than having to develop a completely new program in Fortran, for example.

A group of standardized functions, each designated by an associated function block, may be stored in a control library. A designer equipped with such a library can design process control solutions by interconnecting, on a computer display screen, various functions or elements selected with the function blocks to perform particular tasks. The microprocessor or computer associates each of the functions or elements defined by the function blocks with predefined templates stored in the library and relates each of the program functions or elements to each other according to the interconnections desired by the designer. Ideally, a designer could design an entire process control program using graphical views of predefined functions without ever writing one line of code in Fortran or other high-level programming language.

One problem associated with the use of graphical views for process control programming is that existing systems allow only the equipment manufacturer, not a user of this equipment, to create his own control functions, along with associated graphical views, or modify the predefined functions within the provided library.

New process control functions are designed primarily by companies who sell design systems and not by the end users who may have a particular need for a function that is not a part of the standard set of functions supplied by the company. The standardized functions are contained within a control library furnished with the system to the end user. The end user must either utilize existing functions supplied with

the design environment or rely on the company supplying the design environment to develop any desired particular customized function for them. If the designer is asked to modify the parameters of the engineer's view, then all other views using those parameters have to be rewritten and modified accordingly because the function program and view programs are often developed independently and are not part of an integrated development environment. Clearly, such procedure is very cumbersome, expensive, and time-consuming.

Another problem with existing process control systems is a usage of centralized control, typically employing a central controller in a network, executing a program code that is customized for specialized, user-defined control tasks. As a result, the process control systems are typically constrained to a particular size and difficult to adapt over time to arising needs. Similarly, conventional process control systems are inflexible in configuration, often requiring a complete software revision for the entire system when new devices are incorporated. Furthermore, the conventional process control systems tend to be expensive and usually perform on the functions initially identified by a user or a system designer that are only altered or reprogrammed to perform new functions by an expert who is familiar with the entire control system configuration and programming.

A further problem with existing process control systems is that the physical implementation of different systems is highly variable, including control devices and field devices that have a wide range of "intelligence". For example, some field devices, such as valves, motors and regulators, may have no computational or control capability. Other field devices may have a high level of control autonomy. Still other devices may have some computational strength, but not a sufficient amount to accomplish a desired control task.

What is needed is a uniform or universal design environment that can easily be used, not only by a designer or manufacturer but also a user, to customize a control process to the physical constraints of the process, utilizing control capabilities various controllers and devices, supplementing these control capabilities when desired and distributing control functionality flexibly throughout the process control system to meet specific needs for developing process control functions. What is further needed is a personal computer-based process control system that is easily implemented within substantially any size process and which is updated by users, without the aid of the control system designer, to perform new and different control functions by distributing these control functions throughout the control system including all central, intermediate and peripheral levels.

Diagnostic information is one type of information that is useful to monitor and display in a process control system. However with the various types of devices in a process control system, including a wide variety of field devices, diagnostic information is not generally monitored in a consistent manner from one device to the next. Furthermore, important diagnostic information typically relates to the interaction of multiple portions of the control system, for example, the combined operations of a controller and device or multiple devices and controllers. Diagnostic information relating to multiple circuits in a system is typically not handled by existing process control systems.

Diagnostic information is most useful when related to the various control operations that are occurring when the diagnostic information is monitored. Conventional process control systems typically access and display diagnostic information with no relation to the control operations or control schemes that are functioning during diagnostic testing.

5 One problem associated with the use of graphical views for diagnostic displays is that existing systems allow only the equipment manufacturer, not a user of this equipment, to define the diagnostic information to be monitored, along with associated graphical views, or modify the predefined diagnostic functions within the provided library.

10 What is needed is a uniform or universal design environment that can easily be used, not only by a designer or manufacturer but also a user, to customize monitoring and display of diagnostic operations for a variable number and type of devices and components of a process control system. What is further needed is a personal computer-based process control system that includes a flexible diagnostic monitoring and display functionality that is easily implemented within substantially any size process and which is updated by users, without the aid of the control system designer, to monitor and display diagnostic information for various
15 combinations of process field devices.

In a conventional process control system, the local field devices are typically configured in the field, often by individually programming the local field devices using a hand-held field programmer. Individual programming of the field devices is time consuming and inefficient and often leads to incompatibilities between the device configuration and the configuration of other devices and controllers in the process control
20 system since a global view of the system is more difficult to sustain when individual devices are programmed independently. Usage of individual programming devices is inconvenient since multiple different programming devices typically must be used to program respective different field devices.

Furthermore, local device failures, including temporary failures or local power disruptions, interrupt operations of the entire control system, sometimes causing extended downtime since each failing device must
25 be reconfigured locally.

What is needed is a process control system that allows individual field devices to be configured without local, independent programming. What is further needed is a process control system which allows configuration of the global system from a location remote from the local field devices so that a compatible global configuration is achieved while allowing peripheral elements which are configured in a suitable global
30 manner, to operate independently to achieve control functionality.

Configuration of the global system is based on parameters that describe the particular field devices that make up the system. However, the control protocols for communicating with the field devices may be insufficient to convey parameters that are sufficient to configure the system. For example, the system management specification of the Fieldbus protocol defines three states for a device including an

INITIALIZED state, an UNINITIALIZED state, and a system management operational (SM_OPERATIONAL) state. The three defined states are sufficient to describe the behavior of a device from the perspective of the system management, but are not adequate for describing a device from the perspective of either the fieldbus interface or software engineering tools for analyzing, controlling, or displaying the status of a device. This insufficiency is highly notable when configuration involves the operation of commissioning a device that is attached to the Fieldbus link in an UNINITIALIZED state.

What is further needed is a process control system that differentiates between Fieldbus device states to support automatic sensing of devices and online address assignment of devices.

Several control languages have been developed under an IEC-1131 standard which assist a user in implementing a control strategy. These control languages include function blocks, sequential function charts, ladder logic and structured text. Each of these languages is directed to a particular type of user, including control engineers, control system designers, technicians, operators and maintenance workers. These users have widely different levels and areas of experience, training and expertise. Different users typically view control systems from greatly different perspectives and seek a solution to very different problems, expressed in different manners. For example, a control configuration view of a control system designer may be nonsense to a maintenance worker and vice-versa.

What is needed is a user interface which flexibly presents a configuration in a manner that is most understandable and useful to a particular type of user.

Alarm and event information is one type of information that is highly critical to monitor and display in a process control system. However with the various types of devices in a process control system, including a wide variety of field devices, alarm information is not generally monitored in a useful manner. For example, very different urgencies may exist with respect to a particular alarm. Some alarm conditions may be indicative merely that some routine servicing should take place without urgency. Other alarm conditions require immediate attention. Certain devices in the process control system may measure highly critical conditions while other devices monitor much less urgent information. Furthermore, important some alarm conditions may relate to information the interaction of multiple portions of the control system, for example, the combined operations of a controller and device or multiple devices and controllers. Alarm conditions relating to multiple circuits and devices in a system are typically not handled by existing process control systems.

Alarm and event information is most useful when related to the various control operations that are occurring when the conditions are monitored. Conventional process control systems typically access and display alarm information with no relation to the control operations or control schemes that are functioning during diagnostic testing. Conventional process control systems generally do not have a consistent system for setting priority of different alarm conditions and events.

One problem associated with the use of graphical views for alarm and event displays is that existing systems allow only the equipment manufacturer, not a user of this equipment, to define the alarms and events to be monitored, along with associated graphical views, or modify predefined event priorities. Different types of users may need to visualize different aspects of the process control system. For example, some users have a capability to change only some operating aspects of the control system. These users should have access to condition information which they can control while for other events that may be controlled by another user, alarm information is not urgently needed.

What is needed is a uniform or universal design environment that can easily be used, not only by a designer or manufacturer but also a user, to prioritize display of alarm and event information.

DISCLOSURE OF INVENTION

In accordance with the present invention, a process controller implements smart field device standards and other bus-based architecture standards so that communications and control among devices are performed so that the standard control operations are transparent to a user. The process controller allows attachment to a theoretically and substantially unlimited number and type of field devices including smart devices and conventional non-smart devices. Control and communication operations of the various numbers and types of devices are performed simultaneously and in parallel.

The described design environment enables a process control designer or user to modify a standard process control function or create a unique customized process control function and create the graphical views to be associated with the modified or newly created process control function, all within a common environment. The design environment includes a common interface for both the creation of the function and for its associated engineers, operators, lab and maintenance personnel or other desired users such that when the engineer's function is modified or created, the modification or creation manifests itself in all other graphical views of the function. In addition, the design environment has a common database structure of attributes and methods and the graphics associated with the process control function to allow modified or created process control functions to be represented in whatever graphical methodology that is desired or required by the designer, whether by ladder logic, continuous function block or other design languages required by the various engineer, operator, lab, and maintenance personnel as other desired graphical views.

Many advantages are achieved by the above-described system and operating method. One advantage is that control operations are dispersed throughout the control system, avoiding the inflexibility that arises from centralized control. Another advantage is that the control system is personal-computer (PC) based and, therefore, inexpensive in comparison to mainframe based systems, easily upgraded as additional processes are added to the system, and conveniently operated by multiple users. The PC-based control is further advantageous in allowing user-friendly programming and display platforms such as Windows 95™ and Windows NT™.

In accordance with the present invention, a process controller implements and executes a standard set of function blocks or control functions defined by a standard protocol so that standard-type control is achieved with respect to non-standard-type devices. The process controller enables standard devices to implement the standard set of function blocks and control functions. The process controller implements an overall strategy as if all connected devices are standard devices by usage of a function block as a fundamental building block for control structures. These function blocks are defined to create control structures for all types of devices.

Many advantages are gained by the described system and method. One advantage is that the system is highly uniform, whether attached devices are standard protocol devices or nonstandard devices, thereby improving system reliability. A further advantage is that system development costs are greatly reduced by handling various devices in a uniform manner. Another advantage is that a wide range of different field devices are supported so that intelligent devices utilize the intelligent capabilities and "dumb" devices are controlled by other controllers. An additional advantage is that a software routine performing a particular routine is highly re-usable, improving software reliability.

In accordance with the present invention, a process controller implements an overall, user-developed control strategy in a process control network that includes distributed controller and field devices, such as Fieldbus and non-Fieldbus devices. A user defines the control strategy by building a plurality of function blocks and control modules and downloading or installing user-specified portions of the control strategy into the Fieldbus devices and the non-Fieldbus devices. Thereafter, the Fieldbus devices automatically perform the downloaded portions of the overall strategy independently of other portions of the control strategy. For example in a process control system that includes distributed field devices, controllers and workstations, portions of the control strategy downloaded or installed into the field devices operate independently of and in parallel with the control operations of the controllers and the workstations, while other control operations manage the Fieldbus devices and implement other portions of the control strategy.

The described process control system and operating method has many advantages. One advantage is that the system supplies a uniform, universal design environment for users of many various expertise, experience and training levels to customize a control process to the physical constraints of the process. A further advantage is that the described system uses control capabilities of various controllers and devices, supplementing these control capabilities when desired and distributing control functionality flexibly throughout the process control system as needed. Another advantage is that the process control system is easily based on a personal computer-based design which is easily implemented within substantially any size process and which is updated by users, without the aid of the control system designer, to perform new and different control functions. This flexibility is achieved by distributing control functions throughout the control system including all central, intermediate and peripheral levels.

In accordance with the present invention, a process control system includes a diagnostic monitoring and display functionality for viewing, in a coherent manner, diagnostic information relating to a process that

operates over multiple devices and system components. Although the multiple devices and system components typically encompass widely different device types and operational standards, the process control system incorporates diagnostic information relating to all devices and presents this information to a system user in a uniform manner so that an operating control strategy and the diagnostic information are presented as though all control actions and diagnostic information were performed or generated at a single location. A user-defined diagnostic program is assembled as a set of function blocks and control modules and represented as a set of layers of interconnected control objects identified as modules which include informational structures accessed as attributes. Information is accessed using device hierarchy attribute addressing, supporting direct addressing of I/O signals from modules, bypassing the use of I/O function blocks and avoiding I/O function block behavior.

Many advantages are achieved using the described process control method. One advantage is that the control scheme and the diagnostic monitoring are configured in the system in the same manner, saving system resources and improving system reliability. Another advantage is that configuration of the diagnostics is highly versatile, achieving a wide range of diagnostic behaviors. A further advantage is that the same display objects and procedures are used to display all types of information including configuration information, status information, diagnostics and virtually any other information generated or stored in the system.

In accordance with the present invention, a digital control system automatically senses when a new controller is attached to a network and determines the number and types of I/O Ports that are attached to the new controller. The digital control system formats and displays the I/O Port information upon request by a user. The digital control system program also includes an automatic configuration program that responds to sensing of a new controller by automatically configuring the input/output (I/O) subsystem. The user adds a new controller without setting any physical switches or nodes. A user optionally supplies configuration information for a device into a database, prior to connection of a device. Upon connection of the device, the device is automatically sensed and configured using the database configuration information, without setting of physical switches on the devices.

In accordance with another aspect of the invention, a method of automatically sensing a connection of a controller to a network and incorporating the controller into a network operating system includes the steps of connecting a controller to the network, sending a request from the controller to confirm a network address assignment, the request being accompanied by the controller media access control (MAC) address, a network configuration service receiving the request to confirm and responding. The network configuration service responds by performing the steps of searching a table of configured devices for a matching MAC address and, when the MAC address matches, generating device and network information. The device and network information includes a network address from a device table. When the MAC address does not match, network configuration service generates device and network information including a network address from MAC address-based default information and adds the default information to the device table. When the

MAC address does not match, the network configuration service further performs the step of assigning the connected controller under user control either as a new device added to the device table or as a device configuration previously existing in the device table.

Many advantage are achieved by the described system and method. One advantage is that field devices are programmed from a remote location so that individual field setting of the devices, using a local setting device, is not necessary. Central programmability is highly useful to reduce system management costs and for reducing downtime of a process control system. A further advantage is that configuration of the entire system, rather setting of individual devices, leads to a system in which individual system settings are highly compatible.

In accordance with an aspect of the present invention, a control system controls one or more interconnected devices according to a defined control configuration. The control system automatically senses a device that is connected to the control system but not included in the control configuration definition. The control system supplies initial interconnect information to the connected device sufficient to upload configuration parameters from the device to the control system.

In accordance with a further aspect of the present invention, a digital control system with a predetermined configuration automatically senses the connection to a network of a digital device that is not included in the predetermined configuration. The digital device is assigned temporary address information and placed in a temporary state, called a standby state, in which the digital device supplies information to the digital control system allowing a user to access the digital device including access of device information and configuration parameters. Using the device information and configuration parameters, a user selectively commissions the digital device by assigning a physical device tag, and a device address, and installing a control strategy to the digital device, thereby placing the digital device in an operational state in communication with the digital control system. In the standby state, a user interrogates to determine the type of device that is attached, determines the role of the device in the context of the digital control system, assigns a physical device tag that assigns the determined role to the device, and verifies connection of the device to the network. Also in the standby state, the user initiates other applications applied to the device, including calibration of the device and configuring the device within the overall control scheme of the digital control system.

In accordance with another aspect of the present invention, a control system differentiates between Fieldbus device states beyond the states defined according to the Fieldbus standard specification. The control system sets a physical device tag equal to the device identification (ID) for the devices that do not have tags, while the device is autosensed. A device attached to the Fieldbus link with the physical device tag set equal to the device ID is controlled in the manner of an UNINITIALIZED device.

In accordance with an aspect of the present invention, automatic sensing of field devices is extended beyond a conventional input/ output level to the configuration of Fieldbus devices by a digital control system.

Many advantages are achieved by the described system and method. One advantage is that configuration of a control system is greatly facilitated. The physical connection of a device to the network automatically initiates inclusion of the connected device into the control system. The described system and method advantageously facilitates conformity between the configuration of a network and the physical interconnections of the network that serves as the basis for the configuration. The described system and method assist programming of field devices from a remote location so that individual field setting of the devices, using a local setting device, is not necessary. The system and method support central programmability is highly useful to reduce system management costs and for reducing downtime of a process control system. A further advantage is that configuration of the entire system, rather setting of individual devices, leads to a system in which individual system settings are highly compatible.

In accordance with the present invention, a process control system includes a user interface which supports multiple IEC-1131 standard control languages and user-selection from among the control languages. From a single application routine, a user selects a control language from among a plurality of control languages including, for example, Function Blocks, Sequential Function Charts, Ladder Logic and Structured Text, to implement a control strategy.

In accordance with another aspect of the present invention, a method for configuring a process control environment controlled by a computer system having a processor connected to a display device includes the step of providing a plurality of instructional sections. An instructional section sets forth information relating to configuring the process control environment. The method also includes the steps of selecting a control language editor for defining a process control environment configuration, displaying on the display device a sequence of configuration screen presentations relating to the instruction sections as directed in terms of the selected control language editor, and guiding a user through the configuration of the process control environment via the sequence of configuration screen presentations.

Many advantages are attained by the described system and method. One advantage is that many different users are supported by the system so that users having a wide range of expertise and experience can easily use the system. Furthermore, the system is highly useful for a single user to tailor various aspects of the system using a most appropriate language for a particular system aspect.

In accordance with the present invention, a process control system includes an alarm and event monitoring and display system for which various users of the system can easily prioritize the alarm and event information that is displayed. The alarm and event configuration is highly flexible and is configured by a user to display particular events in a hierarchical manner, as directed by the user. The user sets a desired alarm priority, selecting high importance alarms for more urgent display and annunciation and rendering a lower display status to less urgent events. At log-on, a particular system user is associated with a display configuration for displaying alarm and event information that is pertinent to that user and the process control system is automatically "primed" with current alarms and initiate process information about new alarm and event occurrences.

Many advantages are achieved using the described process control method. One advantage is that alarm information is presented to a user who can best use that information in a manner directed by the user. Another advantage is that a user attains access to the appropriate information automatically, at log-on. A further advantage is that the information stream is "primed" when a user logs on so that pertinent alarm events begin immediate accumulation for that user.

BRIEF DESCRIPTION OF DRAWINGS

The features of the invention believed to be novel are specifically set forth in the appended claims. However, the invention itself, both as to its structure and method of operation, may best be understood by referring to the following description and accompanying drawings.

FIGURES 1A, 1B and 1C illustrate a screen display, a first schematic block diagram and a second schematic block diagram, respectively, process control systems in accordance with a generalized embodiment of the present invention which furnishes a capability to create a new control template and a capability to modify an existing control template for only one view, such as an engineering view.

FIGURE 2 is a schematic block diagram showing the process control environment in a configuration implementation and a run-time implementation.

FIGURE 3 is a block diagram illustrating a user interface for usage with both configuration and run-time models of the process control environment.

FIGURE 4 is a schematic block diagram which depicts a hierarchical relationship among system objects of a configuration model in accordance with an embodiment of the present invention.

FIGURE 5 is a schematic block diagram which depicts a configuration architecture that operates within the hierarchical relationship illustrated in **FIGURE 4**.

FIGURE 6 is a block diagram illustrating an example of an elemental function block, which is one type of system object within the configuration model definition.

FIGURE 7 is a block diagram depicting an example of a composite function block, which is another type of system object within the configuration model definition.

FIGURE 8 is a block diagram illustrating an example of a control module, which is another type of system object within the configuration model definition.

FIGURE 9 is a block diagram showing a module instance, specifically a control module instance, which is created in accordance with the control module definition depicted in **FIGURE 8**.

FIGURE 10 is a flow chart which shows an example of execution of a control module at run-time.

FIGURE 11 is a flow chart which shows an example of a module at a highest layer of a control structure.

FIGURE 12 is a block diagram which illustrates an object-oriented method for installing a process I/O attribute block into a PIO device.

5 **FIGURE 13** is a block diagram depicting an object-oriented method for linking a control module input attribute to a PIO attribute.

FIGURE 14 is a block diagram showing an object-oriented method for linking a control module output attribute to a PIO attribute.

10 **FIGURE 15** is a block diagram showing an object-oriented method for reading values of contained PIO attributes.

FIGURE 16 is a block diagram which illustrates an organization of information for an instrument signal tag.

FIGURE 17 is a flow chart illustrating a method for bootstrap loading a control system throughout a network in the process control environment.

15 **FIGURE 18** is an object communication diagram illustrating a method for creating a device connection for an active, originating side of the connection.

FIGURE 19 is an object communication diagram illustrating a method for creating a device connection for a passive, listening side of the connection.

20 **FIGURE 20** is an object communication diagram illustrating a method for sending request/response messages between devices.

FIGURE 21 is an object communication diagram illustrating a method of downloading a network configuration.

FIGURE 22 is a pictorial view of a front-of-screen display which illustrates a flowchart of the operations of a diagnostic display routine.

25 **FIGURE 23** is an object communication diagram illustrating a method for one device to check whether another device exists on a network.

FIGURE 24 is an object communication diagram illustrating a method for requesting device communications diagnostics.

FIGURE 25 is an object communication diagram illustrating a method for requesting device connection communications diagnostics.

FIGURE 26 illustrates a method for automatically sensing and incorporating a controller/multiplexer into a run-time system.

5 **FIGURE 27** is a flow chart illustrates steps of an automatic configuration routine for configuring a physical I/O device.

FIGURE 28 is a pictorial view of a front-of-screen display for a graphical user interface (GUI) displaying a system configuration.

FIGURE 29 is a state transition diagram illustrating various states of a field device.

10 **FIGURE 30** is a flow chart illustrating a first operation of commissioning a new device.

FIGURE 31 is a flow chart illustrating a second operation of commissioning an unbound.

FIGURE 32 is a flow chart illustrating a third operation of decommissioning a device.

FIGURE 33 is a flow chart illustrating a fourth operation of attaching a commissioned device without enablement of operational powerup.

15 **FIGURE 34** is a flow chart illustrating a fifth operation of replacing a device.

FIGURE 35 is a flow chart illustrating a sixth operation of attaching an UNRECOGNIZED device.

FIGURE 36 is a flow chart illustrating a seventh operation of decommissioning an unrecognized device.

20 **FIGURE 37** is a flow chart illustrating an eighth operation of placing a decommissioned device in a standby condition.

FIGURE 38 is a schematic block diagram which illustrates a program structure of a process control configuration program for defining a process configuration using a plurality of control languages.

25 **FIGURES 39A through 39E** are multiple screen presentations showing configuration, selection and choice screens that are invoked by a configuration program during operation of a configuration operation using a functional block control language and a sequential function chart control language.

FIGURE 40 is an object model showing object relationships of various objects for handling alarm and event functions.

FIGURE 41 is a state transition diagram which depicts alarm attribute states.

FIGURE 42 is a context diagram showing a context for defining an alarm event with respect to a control module.

FIGURE 43 is an object communication diagram illustrating a method for performing an attribute write operation that evokes an "in alarm" status.

MODES FOR CARRYING OUT THE INVENTION

Referring to FIGURE 1A, a control system is shown. In general, the system 1 includes a main processing device, such as personal computer 2, that is connected to a local area network ("LAN") 3 via a local area network card. Although any local area network protocol may be used, a non-proprietary ethernet protocol is beneficial in many applications because it allows for communications with the local area network 3. The local area network 3 is dedicated to carrying control parameters, control data and other relevant information concerned in the process control system. As such, the LAN 3 may be referred to as an area controlled network or ACN 3. The ACN 3 may be connected to other LANs for sharing information and data via a hub or gateway without affecting the dedicated nature of ACN 3.

In accordance with standard ethernet protocol, a plurality of physical devices may be connected to the ACN 3 at various "nodes." Each physical device connected to the ACN 3 is connected at a node and each node is separately addressable according the LAN protocol used to implement ACN 3.

To establish a redundant system, it may be desirable to construct ACN 3 from two or more ethernet systems such that the failure of a single ethernet or LAN system will not result in the failure of the entire system. When such "redundant ethernet" are used the failure of one ethernet LAN can be detected and an alternate ethernet LAN can be mapped in to provide for the desired functionality of ACN 3.

The main personal computer ("PC") A forms a node on the ACN 3. The PC 2 may, for example, be a standard personal computer running a standard operating system such as Microsoft's Window NT system. Main PC 2 is configured to generate, in response to user input commands, various control routines that are provided via the ACN 3 to one or more local controllers identified as element 4 and 5 which implement the control strategy defined by the control routines selected and established in main PC 2. Main PC 2 may also be configured to implement direct control routines on field devices such as pumps, valves, motors and the like via transmission across the ACN 3, rather than through a local controller 4 or 5.

Local controllers 4 and 5 receive control routines and other configuration data through the ACN 3 from PC 2. The local controllers then generate signals of various types to various field devices (such as pumps, motors, regulator valves, etc.) 6 through 15 which actually implement and perform physical steps in the field to implement the control system established by the routines provided by PC 2.

Two types of field devices may be connected to local controller 4 and 5 including field devices 6 through 10 which are responsive to specific control protocol such as FieldBus, Profibus and the like. As those in the art will appreciate, there are standard control protocols (e.g. FieldBus) according to which specific protocol instructions are provided to a protocol-friendly field devices (e.g., a Fieldbus field devices) will cause a controller located within the field device to implement a specific function corresponding to the protocol function. Accordingly, field devices 6 through 11 receive protocol specific (e.g., FieldBus) control commands from either the local controllers 4 and 5 or the personal computer 2 to implement a field device-specific function.

Also connected to local controllers 4 and 5 are non-protocol field devices 12 through 15, which are referred to as non-protocol because they do not include any local processing power and can respond to direct control signals. Accordingly, field devices 12 through 15 are not capable of implementing functions that would be defined by specific control protocol such as the FieldBus control protocol.

Functionality is supplied to allow the non-protocol field devices 12 through 15 to operate as protocol-friendly (e.g., FieldBus specific) devices 6 through 11. Additionally, this same functionality allows for the implementation of the protocol-specific control routines to be distributed between the local field devices 6 through 11, the local controllers 4 and 5 and the personal computer 2.

The distribution of protocol-specific control routines is illustrated in more detail in **FIGURE 1B**. **FIGURE 1B** refers to one portion of the system shown in **FIGURE 1A**, specifically the personal computer 2, the ethernet 3, local controller 4, a smart field device 6 and a dumb device 12, in greater detail.

Personal computer 2 includes program software routines for implementing standard functional routines of a standard control protocol such as the FieldBus protocol. Accordingly, personal computer 2 is programmed to receive FieldBus commands and to implement all of the functional routines for which a local field device having Fieldbus capabilities could implement. The ability and steps required to program personal computer 2 to implement FieldBus block functionality will be clearly apparent to one of ordinary skill in the art.

Connected to personal computer 2 by the ethernet 3 is local controller 4. Local controller 4 includes a central processing unit connected to a random access memory which provides control signals to configure the central processing unit to implement appropriate operational functions. A read only memory is connected to the random access memory. The read only memory is programmed to include control routines which can configure the central processing unit to implement all of the functional routines of a standard control protocol such as FieldBus. Personal computer 2 sends signals through ethernet 3 to the local controller 4 which causes one, more or all of the programmer routines in the read only memory to be transferred to the random access memory to configure the CPU to implement one, more or all of the standard control protocol routines such as the FieldBus routines.

The smart field device 6 includes a central processing unit which implements certain control functions. If the device is, for example, a FieldBus device then the central processing unit associated with the field device 6 is capable of implementing all of the FieldBus functionality requirements.

Because the local controller 4 has the ability to implement FieldBus specific controls, controller 4 operates so that non-protocol device 12 acts and is operated as a FieldBus device. For example, if a control routine is running either in personal computer 2 or on the CPU of local controller 4, that control routine can implement and provide FieldBus commands to FieldBus device 6 and non-protocol device 12, operating as a Fieldbus device. Since field device 6 is a FieldBus device, device 6 receives these commands and thereby implements the control functionality dictated by those commands. Non-protocol device 12, however, works in conjunction with the central processing unit of local controller 4 to implement the FieldBus requirements such that the local controller in combination with the field device implements and operates FieldBus commands.

In addition to allowing non-FieldBus device 12 to act and operate as a FieldBus device, the described aspect allows for distribution of FieldBus control routines throughout the system 1 shown in **FIGURE 1A**. For example, to the extent that a control routine initially requests field device 6 to implement more than one FieldBus control routine, the system 1 allows for control to be divided between the local controller 4 and the local controller 5 such that a portion of the FieldBus control routines are being implemented by local controller 5 and other FieldBus routines are implemented by the use of the FieldBus routines stored on local controller 4. The division of FieldBus routine implementation may allow for more sophisticated and faster control and more efficient utilization of the overall processing power of the system. Still further, the fact that personal computer 2 has the ability to implement FieldBus control routines, the FieldBus routines are further distributed between the local controller 4 and the personal computer 2. In this manner, the system allows personal computer 2 to implement one or all of the FieldBus routines for a particular control algorithm.

Still further, the system allows for the implementation of FieldBus controls to a non-FieldBus device connected directly to the ethernet 3 through use of the FieldBus control routines stored on personal computer 2 in the same manner that FieldBus routines are implemented on non-FieldBus device 12 through use on the FieldBus routines stored on local controller 4.

A process control environment 100 is shown in **FIGURE 1C** and illustrates a control environment for implementing a digital control system, process controller or the like. The process control environment 100 includes an operator workstation 102, a laboratory workstation 104, and an engineering workstation 106 electrically interconnected by a local area network ("LAN") 108 for transferring and receiving data and control signals among the various workstations and a plurality of controller/multiplexers 110. The workstations 102, 104, 106 are shown connected by the LAN 108 to a plurality of the controller/multiplexers 110 that electrically interface between the workstations and a plurality of processes 112. In multiple various

embodiments, the LAN 108 includes a single workstation connected directly to a controller/multiplexer 110 or alternatively includes a plurality of workstations, for example three workstations 102, 104, 106, and many controller/multiplexers 110 depending upon the purposes and requirements of the process control environment 100. In some embodiments, a single process controller/multiplexer 110 controls several different processes 112 or alternatively controls a portion of a single process.

In the process control environment 100, a process control strategy is developed by creating a software control solution on the engineering workstation 106, for example, and transferring the solution via the LAN 108 to the operator workstation 102, lab workstation 104, and to controller/multiplexer 110 for execution. The operator workstation 102 and lab workstation 104 supply interface displays to the control/monitor strategy implemented in the controller/multiplexer 110 and communicates to one or more of the controller/multiplexers 110 to view the processes 112 and change control attribute values according to the requirements of the designed solution. The processes 112 are formed from one or more field devices, which may be smart field devices or conventional (non-smart) field devices. The process 112 is illustratively depicted as two Fieldbus devices 132, a HART (highway addressable remote transducer) device 134 and a conventional field device 136.

In addition, the operator workstation 102 and lab workstation 104 communicate visual and audio feedback to the operator regarding the status and conditions of the controlled processes 112. The engineering workstation 106 includes a central processing unit (CPU) 116 and a display and input/output or user-interface device 118 such as a keyboard, light pen and the like. The CPU 116 typically includes a dedicated memory 117. The dedicated memory 117 includes a digital control system program (not shown) that executes on the CPU 116 to implement control operations and functions of the process control environment 100 shown in FIGURE 1C. The operator workstation 102, the lab workstation 104 and other workstations (not shown) within the process control environment 100 include at least one central processing unit (not shown) which is electrically connected to a display (not shown) and a user-interface device (not shown) to allow interaction between a user and the CPU. In one embodiment, the process control environment 100 includes workstations implemented using a Motorola 68040 processor and a Motorola 68360 communications processor running in companion mode with the 68040 with primary and secondary ethernet ports driven by the 68360 processor (SCC1 and SCC3 respectively).

The process control environment 100 also includes a template generator 124 and a control template library 123 which, in combination, form a control template system 120. A control template is defined as the grouping of attribute functions that are used to control a process and the methodology used for a particular process control function, the control attributes, variables, inputs, and outputs for the particular function and the graphical views of the function as needed such as an engineer view and an operator view.

The control template system 120 includes the control template library 123 that communicates with the template generator 124. The control template library 123 contains data representing sets of predefined or

existing control template functions for use in process control programs. The control template functions are the templates that generally come with the system from the system designer to the user. The template generator 124 is an interface that advantageously allows a user to create new control template functions or modify existing control template functions. The created and modified template functions are selectively stored in the control template library 123.

The template generator 124 includes an attributes and methods language generator 126 and a graphics generator 128. The attributes and methods language generator 126 supplies display screens that allow the user to define a plurality of attribute functions associated with the creation of a new control template function or modification of a particular existing control template function, such as inputs, outputs, and other attributes, as well as providing display screens for enabling the user to select methods or programs that perform the new or modified function for the particular control template. The graphics generator 128 furnishes a user capability to design graphical views to be associated with particular control templates. A user utilizes the data stored by the attributes and methods language generator 126 and the graphics generator 128 to completely define the attributes, methods, and graphical views for a control template. The data representing the created control template function is generally stored in the control template library 123 and is subsequently available for selection and usage by an engineer for the design of process control solutions.

The process control environment 100 is implemented using an object-oriented framework. An object-oriented framework uses object-oriented concepts such as class hierarchies, object states and object behavior. These concepts, which are briefly discussed below, are well known in the art. Additionally, an object-oriented framework may be written using object-oriented programming languages, such as the C++ programming language, which are well-known in the art, or may be written, as is the case with the preferred embodiment, using a non-object programming language such as C and implementing an object-oriented framework in that language.

The building block of an object-oriented framework is an object. An object is defined by a state and a behavior. The state of an object is set forth by fields of the object. The behavior of an object is set forth by methods of the object. Each object is an instance of a class, which provides a template for the object. A class defines zero or more fields and zero or more methods.

Fields are data structures which contain information defining a portion of the state of an object. Objects which are instances of the same class have the same fields. However, the particular information contained within the fields of the objects can vary from object to object. Each field can contain information that is direct, such as an integer value, or indirect, such as a reference to another object.

A method is a collection of computer instructions which can be executed in CPU 116 by computer system software. The instructions of a method are executed, i.e., the method is performed, when software requests that the object for which the method is defined perform the method. A method can be performed by any object that is a member of the class that includes the method. The particular object performing the

method is the responder or the responding object. When performing the method, the responder consumes one or more arguments, i.e., input data, and produces zero or one result, i.e., an object returned as output data. The methods for a particular object define the behavior of that object.

Classes of an object-oriented framework are organized in a class hierarchy. In a class hierarchy, a class inherits the fields and methods which are defined by the superclasses of that class. Additionally, the fields and methods defined by a class are inherited by any subclasses of the class. I.e., an instance of a subclass includes the fields defined by the superclass and can perform the methods defined by the superclass. Accordingly, when a method of an object is called, the method that is accessed may be defined in the class of which the object is a member or in any one of the superclasses of the class of which the object is a member. When a method of an object is called, process control environment 100 selects the method to run by examining the class of the object and, if necessary, any superclasses of the object.

A subclass may override or supersede a method definition which is inherited from a superclass to enhance or change the behavior of the subclass. However, a subclass may not supersede the signature of the method. The signature of a method includes the method's identifier, the number and type of arguments, whether a result is returned, and, if so, the type of the result. The subclass supersedes an inherited method definition by redefining the computer instructions which are carried out in performance of the method.

Classes which are capable of having instances are concrete classes. Classes which cannot have instances are abstract classes. Abstract classes may define fields and methods which are inherited by subclasses of the abstract classes. The subclasses of an abstract class may be other abstract classes; however, ultimately, within the class hierarchy, the subclasses are concrete classes.

All classes defined in the disclosed preferred embodiment, except for mix-in classes which are described below, are subclasses of a class, Object. Thus, each class that is described herein and which is not a mix-in class inherits the methods and fields of class Object.

The process control environment 100 exists in a configuration model or configuration implementation 210 and a run-time model or run-time implementation 220 shown in FIGURE 2. In the configuration implementation 210, the component devices, objects, interconnections and interrelationships within the process control environment 100 are defined. In the run-time implementation 220, operations of the various component devices, objects, interconnections and interrelationships are performed. The configuration implementation 210 and the run-time implementation 220 are interconnected by downloading. The download language creates system objects according to definitions supplied by a user and creates instances from the supplied definitions. Specifically, a completely configured Device Table relating to each device is downloaded to all Workstations on startup and when the Device Table is changed. For controller/multiplexers 110, a downloaded Device Table only includes data for devices for which the controller/multiplexer 110 is to initiate communications based on remote module data configured and used in the specific controller/ multiplexer 110. The Device Table is downloaded to the controller/ multiplexer 110

when other configuration data is downloaded. In addition to downloading definitions, the download language also uploads instances and instance values. The configuration implementation 210 is activated to execute in the run-time implementation 220 using an installation procedure. Also, network communications parameters are downloaded to each device when configuration data are downloaded and when a value is changed.

The process control environment 100 includes multiple subsystems with several of the subsystems having both a configuration and a run-time implementation. For example, a process graphic subsystem 230 supplies user-defined views and operator interfacing to the architecture of the process control environment 100. The process graphic subsystem 230 has a process graphic editor 232, a part of the configuration implementation 210, and a process graphic viewer 234, a portion of the run-time implementation 220. The process graphic editor 232 is connected to the process graphic viewer 234 by an intersubsystem interface 236 in the download language. The process control environment 100 also includes a control subsystem 240 which configures and installs control modules and equipment modules in a definition and module editor 242 and which executes the control modules and the equipment modules in a run-time controller 244. The definition and module editor 242 operates within the configuration implementation 210 and the run-time controller 244 operates within the run-time implementation 220 to supply continuous and sequencing control functions. The definition and module editor 242 is connected to the run-time controller 244 by an intersubsystem interface 246 in the download language. The multiple subsystems are interconnected by a subsystem interface 250.

The configuration implementation 210 and the run-time implementation 220 interface to a master database 260 to support access to common data structures. Various local (non-master) databases 262 interface to the master database 260, for example, to transfer configuration data from the master database 260 to the local databases 262 as directed by a user. Part of the master database 260 is a persistent database 270. The persistent database 270 is an object which transcends time so that the database continues to exist after the creator of the database no longer exists and transcends space so that the database is removable to an address space that is different from the address space at which the database was created. The entire configuration implementation 210 is stored in the persistent database 270.

The master database 260 and local databases 262 are accessible so that documentation of configurations, statistics and diagnostics are available for documentation purposes.

The run-time implementation 220 interfaces to the persistent database 270 and to local databases 262 to access data structures formed by the configuration implementation 210. In particular, the run-time implementation 220 fetches selected equipment modules, displays and the like from the local databases 262 and the persistent database 270. The run-time implementation 220 interfaces to other subsystems to install definitions, thereby installing objects that are used to create instances, when the definitions do not yet exist, instantiating run-time instances, and transferring information from various source to destination objects.

Device Tables are elements of the configuration database that are local to devices and, in combination, define part of the configuration implementation 210. A Device Table contains information regarding a device in the process control environment 100. Information items in a Device Table include a device ID, a device name, a device type, a PCN network number, an ACN segment number, a simplex/
5 redundant communication flag, a controller MAC address, a comment field, a primary internet protocol (IP) address, a primary subnet mask, a secondary IP address and a secondary subnet mask.

Referring to **FIGURE 3**, a block diagram illustrates a user interface 300 for usage with both the configuration and run-time models of the process control environment 100. Part of the user interface 300 is the Explorer™ 310, an interfacing program defined under the Windows NT™ operating system which
10 features a device-based configuration approach. Another part of the user interface 300 is a module definition editor 320 for interfacing using a control-based configuration approach.

The Explorer™ 310 is operated by a user to select, construct and operate a configuration. In addition, the Explorer™ 310 supplies an initial state for navigating across various tools and processors in a network. A user controls the Explorer™ 310 to access libraries, areas, process control equipment and
15 security operations. **FIGURE 3** illustrates the relationship between various tools that may be accessed by a task operating within the process control environment 100 and the relationship between components of the process control environment 100 such as libraries, areas, process control equipment and security. For example, when a user selects a "show tags" function from within an area, a "tag list builder" is displayed, showing a list of control and I/O flags. From the tag list builder, the user can use an "add tag" function to
20 add a module to a list, thereby invoking a "module editor".

Referring to **FIGURE 4**, a schematic block diagram illustrates a hierarchical relationship among system objects of a configuration model 400. The configuration model 400 includes many configuration aspects including control, I/O, process graphics, process equipment, alarms, history and events. The configuration model 400 also includes a device description and network topology layout.

The configuration model hierarchy 400 is defined for usage by a particular set of users for visualizing system object relationships and locations and for communicating or navigating maintenance information among various system objects. For example, one configuration model hierarchy 400, specifically a physical plant hierarchy, is defined for usage by maintenance engineers and technicians for visualizing physical plant relationships and locations and for communicating or navigating maintenance information
30 among various instruments and equipment in a physical plant. An embodiment of a configuration model hierarchy 400 that forms a physical plant hierarchy supports a subset of the SP88 physical equipment standard hierarchy and includes a configuration model site 410, one or more physical plant areas 420, equipment modules 430 and control modules 440.

The configuration model hierarchy 400 is defined for a single process site 410 which is divided into
35 one or more named physical plant areas 420 that are defined within the configuration model hierarchy 400.

The physical plant areas 420 optionally contain tagged modules, each of which is uniquely instantiated within the configuration model hierarchy 400. A physical plant area 420 optionally contains one or more equipment modules 430. An equipment module 430 optionally contains other equipment modules 430, control modules 440 and function blocks. An equipment module 430 includes and is controlled by a control template that is created according to one of a number of different graphical process control programming languages including continuous function block, ladder logic, or sequential function charting ("SFC"). The configuration model hierarchy 400 optionally contains one or more control modules 440. A control module 440 is contained in an object such as a physical plant area 420, an equipment module 430 or another control module 440. A control module 440 optionally contains objects such as other control modules 440 or function blocks. The control module 440 is thus a container class, having instances which are collections of other objects. The control module 444 is encapsulated so that all of the contents and the implementation of the methods of the control module are hidden.

Referring to FIGURE 5, a schematic block diagram shows a configuration architecture 500 that operates within the configuration model hierarchy 400 illustrated in FIGURE 4. The configuration architecture 500 includes a several objects and classes at multiple levels of abstraction. At an organizational level of abstraction 510, the configuration architecture 500 includes a site class 512 which contains "named" objects and classes within the configuration architecture 500. Named objects and classes are definitions, display components such as screens and graphics and other items. The named objects and classes include function blocks, user accounts, modules, plant areas, events, libraries and other site-wide information. Examples of named items are block definitions, equipment module definitions, control module definitions, plant area names and the like.

At a primitive level of abstraction 520, the configuration architecture 500 includes primitives that define the interfaces to functions within the configuration architecture 500, including hard-coded functions such as "+". The primitive level of abstraction 520 includes the classes of functions 522 and parameters 524. Functions 522 are operational functions at the lowest level of abstraction in the configuration architecture 500. Functions 522 are typically coded in the C or C++ languages. In one embodiment of the configuration architecture 500, the full set of implemented function blocks 522 are primitives. Objects and classes at the primitive level of abstraction 520 are defined throughout the site class 512. Parameters 524 are classes and objects at the lowest level of abstraction in the configuration architecture. Parameters 524 include integer numbers, real numbers, vectors, arrays and the like. Attribute values are mapped into parameters 524 for usage within a function block 522. In one embodiment, function blocks 522 at the primitive level of abstraction 520 include the function block primitives listed in TABLE I, as follows:

TABLE I. Function Blocks

| Function Block | Description/Comments |
|----------------|---|
| Action | Handles simple assignment statements using A defined expression capability. |
| ADD | Simple Add function with extensible inputs. |

| Function Block | Description/Comments |
|----------------|--|
| AI | FF Standard Analog Input |
| AI Lite | A scaled back version of the FF analog input. |
| AI HART | The FF Standard Analog Input with some extra ability to handle HART devices. |
| AND | Simple And function with extensible inputs. |
| AO | FF Standard Analog Output. (From FF standard specification) |
| Arithmetic | FF Standard Arithmetic Block. (From FF standard specification) |
| BDE_TRIGGER | Simple bi-directional edge trigger. |
| BIASGAIN | FF Standard Bias/Gain. (From FF standard specification) |
| CALC/LOGIC | Advanced calculation and logic block that has its own language as well as the ability to handle simple ST (1131). It has extensible inputs, extensible outputs, and the ability to create temporary variables. |
| Condition | Handles simple condition statements using The defined expression capability. |
| Counter | Simple up/down counter that handles several different Accumulation methods. |
| CTLSEL | FF Standard Control Selector. (From FF standard specification) |
| DI | FF Standard Discrete Input. (From FF standard specification) |
| DI Lite | A scaled back version of the FF discrete input. |
| DIVIDE | Simple Divide. |
| DO | FF Standard Discrete Output. (From FF standard specification) |
| DT | FF Standard Deadtime with advanced control research implemented. (From FF standard specification) |
| DtoI | A boolean fan in that converts up to 16 discrete inputs to a 16-bit integer value. Also has some special abilities for capturing input patterns. |
| FILT | Simple filter. |
| H/L MON LIMIT | Simple high/low signal monitor and limiter. |
| INTEGRATOR | FF Standard Integrator block. (From FF standard specification) |
| ItoD | Boolean fan-out. Takes a 16-bit integer and translates it into 16 discrete outputs. |
| L/L | FF Standard LeadLag with 2 additional types of equations to select. (From FF standard specification) |
| LOOP | An I/O and control block with the abilities of AI, PID, and AO rolled into one block. |
| LOOPD | An I/O and control block with the abilities of DI, Device Control, and DO rolled into one block. |
| MAN | FF Standard Manual Loader. (From FF standard specification) |
| MULTIPLEX | Simple multiplexor with extensible inputs. |
| MULTIPLY | Simple multiply with extensible inputs. |
| NDE_TRIGGER | Simple negative edge trigger. |
| NOT | Simple not. |

| Function Block | Description/Comments |
|----------------|---|
| OFF_DELAY | Simple off-delay timer. |
| ON_DELAY | Simple on-delay timer. |
| OR | Simple logical or with extensible inputs. |
| P/PD | FF Standard P/PD. (From FF standard specification) |
| PDE_TRIGGER | Simple positive directional edge trigger. |
| PERIOD | Simple monitor that triggers when an input is true for a specified period |
| PI | FF Standard Pulse Input. (From FF standard specification) |
| PID | FF Standard PID with many additions including the ability to choose algorithm type, form, and structure. (From FF standard specification) |
| RAMP | Simple ramp generator. |
| RATELIM | Simple rate limiter generator. |
| RATIO | FF Standard Ratio block. (From FF standard specification) |
| RETENTIVE | Simple retentive timer. |
| RS | Simple reset dominant flip-flop. |
| RUNAVE | Simple running average calculator. |
| SCALER | Simple scaler. |
| SIGGEN | Generates square waves, sin waves, random waves, or any combination of the three. |
| SIGNALCHAR | FF Standard Signal Characterizer. (From FF standard specification) |
| SIGSEL | Simple signal selector. |
| SPLITTER | FF Standard Splitter. (From FF standard specification) |
| SR | Simple set dominant flip-flop. |
| SUBTRACT | Simple subtract block. |
| TP | Simple timed pulse block. |
| TRANSFER | Simple transfer block. |
| XOR | Simple exclusive or block. |

At a definition and usage level of abstraction 530, the configuration architecture 500 includes definitions 532 and usages. Definitions 532 and usages, in combination, define the algorithm and the interface for objects including function blocks, control modules, equipment modules, links and attributes.

- 5 The definitions 532 define algorithms and interfaces for function blocks, modules, links and attributes. Usages are objects and classes at the definition and usage level of abstraction 530 that represent the usage of one definition within another.

- At an instance level of abstraction 540, the configuration architecture 500 includes instances, which are "tagged" items within the configuration. Plant areas 542, modules 544, attributes 546, and PIO blocks 548 are tagged instances. Instances are defined according to definitions 532. A plant area 542 represents a geographical or logical segmentation of a process site class 512. All objects and classes at the instance level of abstraction 540 are defined throughout the plant area level so that all module instances have a 0 or 1

association with a plant area 542. To be installed in a run-time system, the module instances must have a 1
association, meaning that the module is viewed as being "contained by" or "scoped" in this context of a plant
area. A module instance 544 is an installable object that is associated to a specific object of plant equipment.
An attribute instance 546 is a visible parameter in a module instance 544, a plant area instance 542 or other
5 device. An attribute instance 546 may be used for an input signal, an output signal, data storage or the like.

At a device level of abstraction 550, the configuration architecture 500 includes devices 552 such as
controllers, smart devices and consoles, and input/output devices (IO) 560 such as a PIO block, and the like,
which represent physical process control equipment in the physical plant. A process input/output (PIO) block
is an abstraction that represents various high density and low density conventional input/output devices
10 including Hart, FieldBus and other input and output devices that are interfaced into the configuration
architecture 500. High or low density relates to the number of channels on an I/O card. For example, 8
channels are typical on a low density card while a high density card may have 32 channels. Devices 552 are
process control equipment in the configuration architecture 500 and include objects such as controllers,
input/output devices, consoles and the like. Input/output devices (IO) 560 are the physical process input and
15 output devices in the configuration architecture 500.

A smart device is a field device that is implemented to transmit and receive digital data pertaining to
a device, including data relating to device calibration, configuration, diagnostics and maintenance.
Typically, the smart device is also adapted to transmit a standard analog signal that is indicative of various
information including, for example, a process value measured by a field device. Examples of smart field
20 devices include field devices which follow a HART (highway addressable remote transducer) protocol, a
Fieldbus protocol, a Modbus protocol and a device net protocol.

Various hierarchical relationships among system objects are implemented to facilitate navigation
through the process control environment 100 shown in FIGURE 1C by different users and to accomplish
different tasks. Four different hierarchical relationships are defined including control, control system,
25 operations and physical plant hierarchies. A specific system object may be implemented in multiple
hierarchical systems.

The control hierarchy is a subset of a standard SP88 hierarchy and has system objects including site,
physical area, equipment module, control module and control element objects. The control hierarchy is used
to organize control operations and to define the scope of named objects. A user interacts with the control
30 hierarchy on the basis of a site instance, equipment module definitions, control module definitions, a plant
area instance, equipment module instances, control module instances, display module instances, and process
I/O module/block instances, having signal tags.

The control system hierarchy includes operator/configuration stations, host computers, controllers,
I/O devices, smart devices, gateways and the like, which are associated using various network standards
35 including area control network (ACN), process control network (PCN) and other I/O network standards. In

one embodiment, the ACN hardware includes standard 10-base-T ethernet communication ports and a workstation contains standard Ethernet 10-base-T interface cards and software drivers. A user interacts with the control system hierarchy on the basis of a defined site instance, a network definition, a defined network instance, devices, and subsystems such as files, cards, channels, controllers, operation stations, and Fieldbus segments.

The area control network (ACN) includes communication functionality at two levels, a remote object communications (ROC) level and a low level communications level. ROC level controls the interface between the Programmed applications and the ACN communications system. The low level communications support the interface with the TCP/IP sockets and the actual transmission of messages.

Remote Object Communications (ROC) are system operations supporting communication of objects in the process control environment 100 shown in FIGURE 1C and particularly supporting communication between objects whether the objects reside in the same device or in remote devices. The ROC communication level supports communications message services including request/response, unsolicited reporting, event/alarm reporting and broadcast message service.

Request/Response is a service by which applications send messages to a remote device and receive a response from the device. Unsolicited Reporting is a service for periodically sending updated data to a remote device. Unchanged data is not reported. Event/Alarm Reporting is a guaranteed delivery message service which is used for the transmission of events, alarms and other vital information for delivery to a remote device. The broadcast message service is used to send messages to all Program application devices on the communications network. The ROC level also sets communications policies for the communications subsystem. This means that it is responsible for managing what message get sent and when as well as how incoming messages are processed. Communications flow control will also be the responsibility of the ROC portion.

Low level communications support is included for device connection management, ACN redundancy and communications systems diagnostics. Device connection management establishes a communications connection between two devices and manages the transmission of messages between the two devices. ACN Redundancy handles the detection of communications link failures, controls the switch from one link to another and tracks the status of communication links between a host device and connected remote devices. Communications subsystem diagnostics tracks communication integrity and statistical information, responds to requests for communications diagnostic data.

Device connection management in an ACN communications system supports both UDP and TCP type device connections. UDP connections are used for normal real time data transfers between devices. TCP connections are used for special applications using a streaming protocol such as file transfers, device flash downloads, and the like. Communications between devices is managed by a Device Connection Object. The

Device Connection Object transmits data and maintains the status of the communications links between two communicating devices.

All normal device connection message traffic is directed through a single UDP communications port. A Device Connection Object starts the communications system by creating the communication socket associated with this UDP port as well as creating the queues needed for management of the device connection message traffic. The Device Connection Object receives all incoming messages on a Device Connection communications socket and routes messages to the proper device connection instance for processing. The Device Connection Object handles timing functions of device connections, including notifying device connection instances when messages time out waiting to be acknowledged, when communications link checks are due and when device connection resyncs have timed out.

UDP type communications are used for the transfer of real-time data among devices. The remote object communications (ROC) subsystem passes messages to a UDP Device Connection for transmission to a destination device. A pool of message buffers is created on startup of each device. The message pool is used for all data transferred between devices, preventing the communications subsystem from exhausting memory and ensuring that no other task exhausts memory, thereby preventing the communication subsystem from running. Communication flow control is implemented in the Device Connection Object. If the number of message buffers in the communications buffer pool reaches a predefined low level, all remote devices are inhibited from sending messages until the low buffer problem is resolved in the affected device preventing loss of messages.

TCP-type communications are used for applications using a streaming-type protocol such as file transfers and device flash downloads. TCP-type connections are temporary connections established for the duration of the applications needs and terminated once the application has completed a communications task. For reasons of interoperability, compatibility, and availability, a TCP/IP protocol stack is employed. The TCP/IP stack supplies a connection-oriented, byte stream protocol (TCP) and a connectionless, message oriented protocol (UDP). The device connection supports request/response messages, unsolicited data, and event and alarm data between devices. The communication system maintains the device connection through one of two available communications links in the event of a single communications failure, typically a cable fault. Detection of a fault and switch to an alternate communications path transpires in a short, deterministic time span which is less than one second.

The operations hierarchy is defined for a particular set of users, specifically operators and maintenance engineers, generally for the purpose of accessing displays, reports, and other informational items. A user interacts with the operations hierarchy on the basis of a site instance, User Group definitions, a plant area instance, equipment module instances, control module instances, display instances, and report instances.

The physical plant hierarchy is defined for a particular set of users, specifically maintenance engineers and technicians, typically for the purpose of determining physical relationships among objects and navigating maintenance information about plant instruments and equipment. A user interacts with the physical plant hierarchy on the basis of a site instance, a maintenance area instance, a plant area instance, room instances, cabinet instances, node/device instances and display instances.

The system objects that are implemented in the multiple hierarchical systems are arranged into a plurality of subsystems including control, process I/O, control system hardware, redundancy management, event/alarm management, history services, process graphics, diagnostics presentation, user environment, management organization and field management system (FMS) subsystems. The control subsystem includes routines for configuring, installing and executing control modules and equipment modules. The process I/O subsystem is a uniform interface to devices including HART, Fieldbus, conventional I/O and other input/output systems. The control system hardware subsystem defines a control system topology, devices within the topology and capabilities and functions of the devices. The control system hardware subsystem also includes objects and data structures for accessing device level information indicative of status and diagnostics.

The redundancy management subsystem establishes a redundant context between primary and secondary control applications and manages switching in context between the primary and secondary control applications. The redundancy management subsystem also maintains and monitors redundant context diagnostic information including state information and data latency information. Network redundancy is accomplished using two separate Ethernet communications links or networks. The primary communication link is the preferred communications path. The secondary link is only used if the primary has failed. Communications switchovers are performed on a per device basis. For example, if device A is communicating with devices B and C and the primary link to device C fails, device A continues to communicate with device B on the primary link but switches to the secondary link to communicate with device C. Each Ethernet link is a separate, dedicated network having a dedicated set of IP addresses and a subnet mask.

The device connection object manages redundant communications including controlling when to switch to the secondary link and when to switch back to the primary link. Each device in the process control system tracks the communication status of all current links to remote devices by periodically sending link test messages when no other communications is occurring, to check the status of the communications links to each device. Redundancy switchovers are performed on a device connection basis.

The event/alarm management subsystem configures, monitors, and supplies notification of significant system states, acknowledgments and priority calculations. The history services subsystem stores and retrieves process and event information. The process graphics subsystem supplies user-defined views for display and operator interfacing onto the defined system architecture. The diagnostics presentation subsystem furnishes displays of diagnostic information, typically at the request of a user. The user

environment subsystem supplies a user interface, allowing a user to enter commands to control operation of the process control environment 100 shown in FIGURE 1C. The management organization subsystem sets an organizational structure of the process control environment 100 including specification of site, area, primitives, access to user libraries, and location of defined objects and instances. The FMS supplies user interfaces, views, and organization structure for the configuration, installation and monitoring of HART and Fieldbus devices.

A Fieldbus device implements localized control of a process within the process, in contrast to a longer-used and more conventional approach of controlling device functions from a main or centralized digital control system. A Fieldbus device achieves localized control by including small, localized controller/multiplexers 110 which are closely associated with field devices within the Fieldbus device. The small, localized controllers of a Fieldbus implement standardized control functions or control blocks which operate on the field devices within the Fieldbus device and which also operate on other smart field devices that are connected to the Fieldbus device.

Thus, the process control environment 100 implements smart field device standards, such as the Fieldbus H1 standard, Profibus standard, CAN standard and other bus-based architecture standards so that communications and control among devices, particularly Fieldbus devices, are performed so that Fieldbus-type control operations are transparent to a user.

The process control environment 100 allows attachment to a substantially unlimited number and type of field devices including smart devices, such as Fieldbus and HART devices, and conventional non-smart devices. Control and communication operations of the various numbers and types of devices are advantageously performed simultaneously and in parallel.

The process control environment 100 implements and executes a standard set of function blocks or control functions defined by a standard Fieldbus protocol, such as the Fieldbus H1 standard, so that smart-type control is achieved with respect to non-smart-type devices, such as a HART device 134 and a conventional device 136. In addition, the process control environment 100 enables Smart devices to implement the standard set of function blocks and control functions. Advantageously, the process control environment 100 implements an overall strategy as if all connected devices are Smart devices. This implementation is achieved, in part, by the usage of a function block as a fundamental building block for control structures. These function blocks are defined to create control structures for all types of devices. Usage of function blocks as fundamental building blocks is described in FIGURES 6, 7, 8 and 9.

The process control environment 100 implements an overall, user-developed control strategy through the definition of function blocks and control modules by downloading or installing specific portions of the control strategy into smart devices and non-smart devices. Thereafter, the smart devices automatically perform the downloaded portions of the overall strategy independently of other control system operations. For example, the portions of the control strategy downloaded or installed into the devices operate

independently of and in parallel with the control operations of the controller/ multiplexers 110 and the workstations, while other control operations manage the smart devices and implement other portions of the control strategy. In effect, the process control environment 100 implements a control strategy using the controller/ multiplexers 110 within the smart devices.

5 An example of a block diagram defining a function block of the functions 522 shown in **FIGURE 5** is illustrated in **FIGURE 6**. Specifically, **FIGURE 6** depicts an "elemental" function block definition 600 which is defined to contain only primitive objects. The elemental function block definition 600 defines a sum function and includes a "+" primitive 610, a first input attribute 612 which is a first parameter 614 applied to the primitive 610, and a second input attribute 622 which is a second parameter 624 applied to the primitive
10 610. The primitive 610 produces a result that is supplied as an output attribute 630. In this example, the elemental function block definition 600 is a block definition that is created and named SUM.

A second example of a block diagram defining a function block of the functions 522 shown in **FIGURE 5** is illustrated in **FIGURE 7**. Specifically, **FIGURE 7** depicts a "composite" function block
15 definition 700 which is defined to contain one or more elemental function blocks 600 and, optionally, one or more primitive objects. The composite function block definition 700 defines a composite sum function and includes a first SUM elemental function block 710 and a second SUM elemental function block 712, each of which is the same as the SUM elemental function block 600 illustrated in **FIGURE 6**. The composite function block 700 has a first input attribute 720 and a second input attribute 722 which are respective first and second parameters 724 and 726 applied to the first SUM elemental function block 710. The first
20 elemental function block 710 produces a result that is applied to the second SUM elemental function block 712 as a first parameter 730. The composite function block 700 has a third input attribute 728 that is a second parameter 732 applied to the second SUM elemental function block 712. The second SUM elemental function block 712 produces a result that is supplied as an output attribute 734. In this example, the composite function block definition 700 is a block definition that is created and named SUM3.

25 An example of a block diagram defining a control module of the control modules 440 shown in **FIGURE 4** is illustrated in **FIGURE 8**. Specifically, **FIGURE 8** depicts a control module definition 800 which is defined and contains various input attributes 810, elemental function blocks 820, a first SUM3 composite function block 830 and a second SUM3 composite function block 832. The exemplary control module 440 includes five input attributes 810 which are connected to five respective elemental function
30 blocks 820, three of which are parameters applied to the first SUM3 composite function block 830. The first SUM3 composite function block 830 produces a result that is supplied as a parameter to the second SUM3 composite function block 832 in combination with parameters supplied by the remaining two elemental function blocks 820. The second SUM3 composite function block 832 produces a result that is supplied as an output attribute 840. In this example, the control module 800 is a control module definition that is created
35 and named CM1.

Examples of block diagrams defining a module instance of the module instances 544 shown in **FIGURE 5**, specifically a control module instance, are shown in **FIGURE 9**. Control module instances 910 and 920 are created in accordance with the CM1 control module definition so that each control module instance 910 and 920 includes five input attributes 912 and 922, respectively, that correspond to the five input attributes 810 shown in **FIGURE 8**. Each control module instance 910 and 920 also includes one output attribute 914 and 924, respectively, that correspond to the output attribute 840 shown in **FIGURE 8**. In this example, the control module instances 910 and 920 are control module instances that are created and tagged **CALC1** and **CALC2**, respectively.

Using a definition editor, a system user creates and names definitions, such as the **SUM** elemental function block definition, the **SUM3** composite function block definition and the **CM1** control module definition. Then, using a module editor, the system user creates and tags instances, such as the **CALC1** and **CALC2** control module instances.

Referring to **FIGURE 10**, a flow chart shows an example of control module execution at run-time. A run-time program includes a scheduler routine. Scheduler routines are well-known in the computing arts. The scheduler routine requests execution 1010 of a composite control module, for example the composite control module 440 with tag **CM1** shown in **FIGURE 8**. The **CM1** composite control module 440 initiates transfer 1012 of the input attributes 820, causing any associated links, or attribute associations, to transfer 1014. A database definition typically refers to "associations" while a runtime definition relates to "links". In steps 1016 through 1056 the **CM1** composite control module 440 requests each elemental function block 820, first **SUM3** composite function block 830 and second **SUM3** composite block 832 to execute in turn.

Specifically, in step 1016 the **CM1** composite control module 440 requests each elemental function block 820 to execute. The elemental function blocks 820 initiate transfer 1018 of input attributes, for example first input attribute 612 shown in **FIGURE 6**. The input attributes of the elemental function blocks 820 request 1020 loading of values from the links transferred in step 1014. The links copy 1022 values from source attributes to destination attributes. The elemental function blocks 820 execute block algorithms 1024. Upon completion of block algorithm execution, the elemental function blocks 820 initiate transfer of output attributes 1026, for example output attribute 630 shown in **FIGURE 6**.

In step 1028 the **CM1** composite control module 440 requests first **SUM3** composite function block 830 to execute. First **SUM3** composite function block 830 initiates transfer 1030 of input attributes, for example input attributes 722, 724 and 726 shown in **FIGURE 7**, from the elemental function blocks 820. In step 1032, first **SUM3** composite function block 830 requests internal function blocks, for example, the first **SUM** elemental function block 710 and the second **SUM** elemental function block 712 shown in **FIGURE 7**, to execute in turn. First **SUM** elemental function block 710 reads input attributes, executes a block algorithm and sets an output attribute in step 1034. Second **SUM** elemental function block 712 reads input attributes, executes a block algorithm and sets an output attribute in step 1036. First **SUM3** composite function block

830 initiates transfer of output attributes in step 1038. The output attribute of first SUM3 composite function block 830 requests an associated link to copy the value from the output attribute in step 1040.

In step 1042 the CM1 composite control module 440 requests second SUM3 composite function block 832 to execute. Second SUM3 composite function block 832 initiates transfer 1044 of input attributes from the links connected to the first SUM3 composite function block 830 output attributes. In step 1046, second SUM3 composite function block 832 requests internal function blocks, for example, the first SUM elemental function block 710 and the second SUM elemental function block 712 shown in FIGURE 7, to execute in turn. First SUM elemental function block 710 reads input attributes, executes a block algorithm and sets an output attribute in step 1048. Second SUM elemental function block 712 reads input attributes, executes a block algorithm and sets an output attribute in step 1050. Second SUM3 composite function block 832 initiates transfer of output attributes in step 1052. The output attribute of second SUM3 composite function block 832 requests an associated link to copy the value from the output attribute in step 1054.

In step 1056 the CM1 composite control module 440 initiates transfer of output attributes and output attribute 840 requests a link from second SUM3 composite function block 832 to copy the value of the second SUM3 composite function block 832 output attributes. In some embodiments, output function blocks push output values to a user-configured PIO block attribute (not shown). Thus, PIO attributes are "pulled" by function blocks while output function blocks push output values to PIO Block attributes. Similarly, input function blocks pull input attribute values from PIO Block attributes.

A user defines a module control strategy by specifying function blocks that make up control modules and determine the control strategy. The user modifies or debugs a module control strategy by adding, modifying and deleting function blocks, configuring parameters associated with the function blocks and creating a view to new attributes.

By defining function blocks and control modules, a user-defined control strategy, application program or diagnostic program is represented as a set of layers of interconnected control objects identified as modules. A layer of the control strategy includes a set of modules which are interconnected in a user-specified manner. A module typically includes an algorithm for performing a specific function and display components which are used to display information to a user. A module is optionally represented to include a set of input and output connections for connecting to other modules. A module may be considered to be a "black box" which performs a specified function and is connected to other modules via specified input and output connections.

Referring to FIGURE 11, a display screen serves as a flow chart which shows an example of a module or application program LOOPSIM 1060 at a highest layer of a control structure. The illustrated layer of the LOOPSIM 1060 application program includes an input attribute (AIN) module 1062 called AI1, a deadtime module 1064, a proportional, integral, differential (PID) control module 1066, an output attribute (AOUT) module 1068 and a simulate module 1070. Each of the illustrative modules includes named input

connections and output connections which are connected to the other modules via lines. The set of modules, the input connections and the output connections of the set of modules, and the interconnections between modules define the operation of the LOOPSIM 1060 application.

At a lowest level, a module is a set of interconnected function blocks. At higher levels, a module is a set of interconnected submodules which, in turn, may include a further set of submodules. For example, the PID control module 1066 is typically a set of interconnected submodules which perform the different functions included in a PID functionality. The input and output connections of the PID module 1066 are an input connection and an output connection of one or more of the submodules within a next lower layer of the PID module 1066. The submodules in the PID module 1066 optionally include other input and output connections sufficient to define the interconnections between the submodules.

An application, a module or a submodule, at any module level, is optionally modified by a user to perform a slightly different function or to perform the same function in a different manner. Thus, a user optionally modifies the module, thereby modifying the control structure, in a desired manner. Specifically, a user optionally adds input and output connections to modules and extends the input and output connections of a module to a higher level module so customize modules for various applications. For example, a user optionally adds a new input connection or output connection to the PID module 1066 to the "edge" of the PID module 1066 which makes the input connection and output connection appear as input and output connections to the PID module 1066.

The process control environment facilitates the definition and modification of the control structure by furnishing editing operations in a plurality of control languages including IEC-1131 standard languages such as Field Blocks, Sequential Function Charts (SFC), Ladder Logic and Structured Text. Accordingly, different types of users, from different control backgrounds use the different languages to write different modules for implementing the same or different applications.

Control modules are specified to have several advantageous characteristics. Some control modules allow direct access to attributes. For example, some attributes, called "heavy" attributes, support direct (maximum performance) communications. Direct communications are advantageously used for connecting function blocks and Control Modules, supporting event/alarm detection, and high performance trending, for example. Some attributes are created automatically upon definition of a control module with a user having the option to promote or force a parameter to be exposed as an attribute of a Control Module. Other parameters are made accessible through a module, such as a Control Module, an Equipment Module, a PIO Block, or a Device, which contains the parameter but direct communications performance of the attributes does not warrant the overhead incurred in supplying this performance. These parameters are advantageously accessed to supply information relating to control system tuning, debugging and maintenance. In some embodiments, these parameters are accessed by a general purpose parameter browser applications, which use services provided by tagged containers to reveal attributes, invokeable services, and subcomponents within the containers.

Referring to **FIGURE 12**, a block diagram illustrates an object-oriented method for installing a process I/O attribute block into a PIO device through the operation of the control subsystem. A block of defined objects 1110 includes a site object 1112, a controller device 1114, a controller I/O subsystem 1116, a PIO interface device 1118 and a PIO device 1120. Prior to installation of the PIO device, the controller I/O subsystem 1116 is previously created. The PIO device 1120 is also previously created, either by installation or downloading. The block of defined objects 1110 directs a detail pointer 1122 to a list of block definitions 1124 to specify a particular type of object to be created by a create pointer 1126 directing the operation of a create block 1128. The block definitions 1124 includes a PIO input attributes (AIN) block definition either as hardwired or by previous installation. Attributes of the specified object are set by a user through the operation of an editor 1130. Prior to installation of the PIO device, an input attribute (AIN) block 1132 does not exist.

Prior to installing the AIN block 1132, a user creates the PIO device 1120 then sets up initial values for AIN block attributes using the editor 1130. The user also sets a period for view parameter acquisition. The AIN block 1132 is saved and then installed.

Referring to **FIGURE 13**, a block diagram illustrates an object-oriented method for linking a Control Module input attribute to a process I/O attribute. Prior to linking of the control module input attribute to the PIO attribute, the PIO block AIN 1220 is previously installed and the control module 1210 is also installed. The user specifies that a PIOIN attribute 1212 of the control module 1210 is connected to an attribute input process variable PV 1214 and requests that a link be made. A link 1216 is made as the control module finds the PIOIN attribute and returns a corresponding attribute index, locates PIO AIN in a plant area, find the process variable PV attribute and returns a corresponding attribute index, instructs the run-time linker 1216 to create a link with a source at the process variable (PV) 1214 and a destination at the PIOIN attribute 1212, creates the link and connects the link 1216. At end of a download, links are resolved by the linked objects.

Referring to **FIGURE 14**, a block diagram shows an object-oriented method for linking a control module output attribute (AOUT) 1312 attribute to a PIO output attribute (PIOAOUT) 1320. A control module 1310 is previously installed and the control module output attribute (AOUT) 1312 is installed within the control module 1310. The user specifies that the control module output attribute (AOUT) 1312 is connected to the a PIO output attribute (PIOAOUT) 1320. The link is made as the run-time implementation of the control module 1310 is sent a message to form the connection, the control module 1310 finds the AOUT attribute, requests location of the PIOAOUT attribute 1320, creates a link 1322 and connects the AOUT attribute 1312 and the PIOAOUT attribute 1320 to the link 1322.

Referring to **FIGURE 15**, a block diagram shows an object-oriented method for reading values of contained PIO attributes. A PIO block 1410 is previously installed and an output attribute (AOUT) 1412 is previously installed within the PIO block 1410. A user, for example an engineer, requests a detailed view of

the block in which all attribute values are displayed. The detailed display includes one or more sets of display groups, also called view definitions, associated with the PIO block 1410. A proxy is previously established for the PIO Block 1410. A user requests detail for the output attribute (AOUT) 1412. Attribute names and values for the AOUT block are presented by an application program requesting a proxy client routine to access a view, an AOUT proxy client setting a return view definition and creating an attribute proxy object, and the application program requesting the AOUT proxy client to read out values for attributes named with granted privileges. The application program formats and displays the data. Display group parameters are part of an I/O block definition and are, therefore, not configurable. Display groups are defined for attributes. Information is advantageously updated while a PIO block is not linked since display groups and view groups control updating of non-linked PIO attributes associated with a block.

The process control environment 100 shown in **FIGURE 1C** implements an overall strategy as if all connected devices are Fieldbus devices not only by the usage of a function block as a fundamental building block for control structures, but also by implementing an input/output architecture that treats Fieldbus and nonFieldbus devices in the same manner. The fundamental character of the input/output architecture is based on instrument signal tags (ISTs) that furnish user-configurable names for all I/O signals including Fieldbus and nonFieldbus I/O signals.

From the perspective of a user, an IST binds a user-defined name to a signal type, to a specific signal in the I/O subsystem, to a signal path including an attribute and to a set of signal property settings.

ISTs are not installed in the manner of other system objects. Instead, signal properties inherent to the IST tag are combined with I/O Port and I/O Device properties that are made available when an I/O Card is installed. The combination of IST, I/O Port and I/O Device properties furnish information for creating a PIO function block in the run-time system. The signal path from ISTs is included in the script that defines I/O Function Blocks during installation of a module.

To communicate throughout the process control environment 100, an I/O type Function Block uses an I/O reference definition. An IST satisfies the specification for an I/O reference. Conventional I/O devices, such as MTL devices supplied by Measurement Technologies Limited of the United Kingdom, have an IST for each channel. Hart and Fieldbus I/O devices may include an IST for each distinct "I/O signal" on a Port or in a field Device. IST names have system-wide scope and share the name space of Modules, Devices, and Areas. In large systems, ISTs typically correspond to instrument signal names on instrumentation drawings. In small systems, formal instrument drawings may not exist so that no obvious IST names are inferred. Typically, ISTs are automatically generated as cards are configured based on a device hierarchy path representing a controller node, I/O subsystem, card and port so that arbitrary IST names are avoided. Typically most ISTs are created automatically when a new I/O card is defined. For multiple-signal I/O devices, an IST is automatically created for only a single "primary signal". In addition to automatic IST creation, a user may also create ISTs using an "Assign..." menu available from the

Explorer Node/IOsubsys/Port/Device tree with a Port or Device selected or using a "New..." menu available from the Explorer IST tree.

ISTs have a "signal type" property to ensure compatibility between the I/O signal and the I/O Function Block(s) that accesses the I/O signal. Signal type is one of: AIN, AOUT, DIN, DOUT, PCIN, PCOUT. ISTs have a set of "signal-related" attributes specific to the signal type (e.g. EU0 and EU100 for a AIN, MOMENTARY or LATCHED for a DOUT, etc.). All signal sources with the same signal type have the same set of "signal attributes". All other properties of the I/O subsystem objects are held in card, port, or device attributes.

Fully configured ISTs have a fully qualified path to a corresponding signal in the I/O system, e.g. "CON1/IO1/S01/C01/FIELD_VAL". An IST may be created without a defined path defined so that module configuration may be completed before I/O structure details are fully defined. Modules with I/O Function Blocks using ISTs with no defined path may be configured and installed but the run-time system must deal appropriately with missing I/O paths of missing ISTs on I/O Function blocks. A signal source has no more than one IST. Attempts to configure more than one IST with the same path are rejected.

A user may delete an IST, thereby deleting associated signal properties and possibly leaving some unresolvable IST references in I/O Function Blocks. A deleted IST does not affect card/port/device properties with a normal display of the IST on the Port/Device in the Explorer tree indicating no associated IST.

I/O-interface Function Blocks have at least one IST-Reference property. An IST-Reference property is either left blank to indicate that the function block does not connect to a IST, or is designated with a valid IST name. An IST-Reference property in an I/O Function Block is compatible with exactly one IST signal type. For example, the IST-Reference in the AI Function Block has an IST with a signal type "AIN" only. Several I/O Function Blocks are compatible with the same IST signal type.

For compatibility with Fieldbus I/O Function Block definitions, Fieldbus I/O Function Blocks have attributes such as XD_SCALE, OUT_SCALE which overlap with some of the signal properties in ISTs.

When a valid IST-Reference is made, the configured values of these overlapped Function Block attributes are ignored in the Run-time system and the corresponding properties from the IST are used instead. An engineer configuring Fieldbus I/O Function Blocks uses an indication of ignored attributes when a IST reference is in place. Such an indication is typically presented on a display as grayed out and non-editable text with values copied from the IST. The I/O Function Block holds a private setting for the ignored attributes which are typically downloaded and promptly overridden. If the IST-Reference is removed, the setting for these attributes retains utility.

I/O Cards, Ports and Devices are incorporated into a configuration by a user operating a user interface, either the Explorer™ or the Module Definition Editor. The channels on conventional I/O cards are called "ports" and treated as an I/O Port so that special case terminology for conventional I/O is avoided.

The user interface also allows a user to delete I/O Cards, Ports or Devices. Multiple I/O Card types are supported including, for example, 8-chan MTL AI, 8-chan MTL AO, 8-chan MTL DI, 8-chan MTL DO, 4-chan MTL Thermocouple/RTD, 8-chan HART input, 8-chan HART output, and 4-chan Solenoid. Some I/O Card types have a combination of I/O Port types on the same I/O Card. Deletion of an I/O Card deletes all subordinate Ports. Deletion of an I/O Port deletes all subordinate Devices. Deletion of I/O Ports or I/O Devices does not delete related instrument signal tags (ISTs), but the path of the IST path to the associated I/O signal no longer is operable. If another I/O Port or I/O Device is created which has the same path, the IST automatically rebinds to the I/O Port or I/O Device, so long as the signal type is compatible.

A user can initiate the Install of an I/O subsystem, which installs or reinstalls all I/O Cards defined in the Subsystem. The user can initiate the Install of a single I/O Card, which installs the card properties and all properties for subordinate I/O Ports and I/O Devices.

The Explorer™ and the Module Definition Editor configure the I/O subsystem by accessing current signal values, status, and selected properties that are directly addressable as Attributes in the I/O subsystem. The user displays a graphic indicative of the current status of cards, ports, devices, and signal values and status by accessing the respective cards, ports, devices and signal values and status using device hierarchy attribute path addressing (for example, "CON1/IO1/C01/P01/FIELD_VAL").

I/O subsystem attributes are communicated using the physical device path (for example, CON1/IO1/C01/P01/D01/FIELD_VAL) for addressing in communications between devices. Communication of I/O subsystem attributes is advantageously used to transmit attributes from a controller/multiplexer 110 to a workstation 102, 104, 106 as shown in FIGURE 1C for display and from a first to a second controller/multiplexer 110 for virtual I/O handling.

Referring to FIGURE 16, a block diagram illustrates an organization of information for an instrument signal tag (IST). An system IST table 1510 contains information relating to an IST including path information and pointers to a system object. A first pointer 1512 designates a signal type which points to an attribute signal table 1520. A second pointer 1514 designates an entry in the attribute signal table 1520.

Accessing of information using device hierarchy attribute addressing advantageously allows system diagnostic displays to be designed and built for system integration checkout before Control Module work is complete. Device hierarchy attribute addressing also supports direct addressing of I/O signals from Modules, bypassing the use of I/O function blocks and avoiding I/O function block behavior. I/O Card, I/O Port and I/O Device identifiers are generally defined automatically according to slot position information and the like.

Referring to FIGURE 17, a flow chart illustrates a method for bootstrap loading a control system throughout a network in the process control environment 100, including the operations of assigning the controller/ multiplexers 110 to a set of IP Addresses, a node name and other startup information that is not

stored in flash ROMs of a controller/ multiplexer 110. A process 1600 for assigning internet protocol (IP) Addresses to a Controller upon its initial bootup includes the step of associating a MAC address in a Boot server, a Windows NT™ workstation, with a controller/ multiplexer name 1610. The MAC address alone designates the controller/ multiplexer identity. In step 1612, the name of the controller/multiplexer is assigned an arbitrary device ID, and an ACN link number and a PCN network number that are determined by the cable attached to the controller/ multiplexer. In step 1614, an IP address of a device is calculated from the device ID, the ACN link number and the PCN network number. In step 1616, a UDP datagram, which designates default primary and secondary IP addresses that are reserved for booting nodes and includes the controller/ multiplexer MAC address in the UDP user data, is broadcast to a special UDP reserved boot port using the default primary IP address for the source address on the primary interface. In step 1618, the boot server matches the MAC address with the assigned name and IP addresses, and broadcasts the assigned name and IP addresses with an echo of the MAC address to the UDP boot port. By broadcasting, the problem of doing any routing or ARP static entry manipulation is avoided. In step 1620, the controller/ multiplexer receives the datagram, checks the MAC address, and if the MAC address matches, sets the IP addresses and saves the node name and device ID. If the datagram is not received, the procedure is repeated using the secondary interface through the operation of branch step 1622. In step 1624, the controller/ multiplexer, using the new address, sends a message to the boot server saying indicating that the controller/ multiplexer is operational.

In step 1626, a user enters a Device Name, Device MAC Address, ACN Link Number and PCN Network Number. The device ID can be automatically assigned by configuration software. The communications subsystem calculates the devices three IP addresses from the configured ACN Link number, PCN Network Number and the assigned device ID. In step 1628, controller/ multiplexer or I/O card software is flash downloaded over the ACN network by passing messages and S-Record files between devices on the ACN.

Referring to FIGURE 18, an object communication diagram shows a method for creating a device connection for the active, originating side of a connection. An application program in either a workstation or a controller/ multiplexer requests access to an attribute which is contained in another device. A UDP communications connection to the other device is established by the communication services so that the attribute can be accessed. Creation of a device connection spans two separate application programs. The application program which initiates the connection by requesting data located in another device and the Remote Object Communications (ROC) Services application program that actually sends the messages to the other device. If no connection exists when the ROC Services process is ready to send a message to a device, the ROC services create a connection to that device.

Prior to creating the device connection, a device to be connected has a valid Device Table containing the source device, is operating and includes an object **RtDeviceConnection** which monitors messages on the

device connection port. After the device connection is created, a connection is established between the two devices and an **RtDeviceConnection** instance is created in the active device to handle the connection.

In step 1710, an application program sends a message **getContainer** to object **RtSite** which returns the object ID of the module found or created. In step 1712, object **RtSite** sends a **Locate** message to object **RtPlantArea** which locates the module and return its object ID. In step 1714, object **RtSite** sends a **GetDevice** message to object **RtDevice** which returns the object ID of the device containing the module. In step 1716, assuming that a proxy for the remote device does not exist, object **RtDevice** sends a **Create** message to object **RtDeviceProxy**. In step 1718, object **RtDeviceProxy** creates an instance of object **RtDeviceProxy** using template **RtNew**. In step 1720, object **RtDeviceProxy** asks object **RtDeviceConnection** to **GetDeviceConnectionIndex** which returns the index of the device name in the device connection table managed by object **RtDeviceConnection**. In step 1722, object **RtDeviceProxy** registers the pointer to the **RtDeviceProxy** instance for the connected device by sending a **RegisterPointer** message to the object **RtRegistry** and returns the device proxy Object ID to object **RtDevice**. In step 1724, object **RtPlantArea** sends a **Create** message to object **RtModuleProxyClient** to create a proxy client for the remote module. In step 1726, object **RtModuleProxyClient** sends a **Create** message to object **RtModuleProxyServer** to create a proxy server for the module in the remote device. In step 1728, object **RtModuleProxyServer** builds a create proxy server message and asks object **RtRocReqRespService** to **SendRequest** to the remote device. In step 1730, object **RtRocReqRespService** Appends the message to the **Outbound Message Queue** for the ROC Communications Services process to send to the remote device. In step 1732, object **RtRocReqRespService** in the ROC Comm Services process issues a **RemoveFirst** command to the **Outbound Message Queue** and gets the create proxy server message. In step 1734, the **RtRocReqRespService** sends the message by issuing a **sendMsg** command to the **aRtDeviceProxy** instance for the destination device. In step 1736, the **aRtDeviceProxy** instance issues a **GetDeviceConnection** command to **RtDeviceConnection** to get the Object ID for the **RtDeviceConnection** instance for the destination device. Assuming that a device connection does not already exist, in step 1738, object **RtDeviceConnection** performs a **createDeviceConnection**. In step 1740, object **RtDeviceConnection** creates an instance of **RtDeviceConnection** using template **RtNew**. In step 1742, object **RtDeviceConnection** registers the pointer to the **RtDeviceConnection** instance by sending a **RegisterPointer** message to the object **RtRegistry** and returns the device connection Object ID to object **RtDeviceConnection**. In step 1744, object **RtDeviceConnection** sends a **startActiveConnection** message to the **aRtDeviceConnection** instance. The **aRtDeviceConnection** instance performs the necessary steps to establish the connection to the other device. In step 1746, the **RtDeviceProxy** instance issues a **sendMsg** to the **aRtDeviceConnection** instance to send the create server message to the remote device. In step 1748, the **aRtDeviceConnection** instance sends the message to the remote device over the newly created connection.

Referring to **FIGURE 19**, an object communication diagram shows a method for creating a device connection for the passive, listening side of a connection. A request to establish a device connection is

received from another workstation or controller/ multiplexer. The communications services establishes a UDP communications connection with the requesting device.

Previous to creation of the connection, a device to be connected to is operating and contains an object **aRtDeviceConnection** which is ready to establish a connection. Object **RtDevice Connection** exists in the device and is listening for input messages in the form of a sync request. After the connection is created, a connection is established between the two devices and an **RtDeviceConnection** instance is created in the passive device to handle the connection.

In step 1810, object **RtDeviceConnection** receives a sync request message from a remote device. In step 1812, object **RtDeviceConnection** sends a **Create** message to object **RtDeviceConnection** to create a connection to the requesting device. Assuming that a device connection does not already exist, object **RtDeviceConnection** performs a **createDeviceConnection** in step 1814. In step 1816, object **RtDeviceConnection** creates an instance of **RtDeviceConnection** using template **RtNew**. In step 1818, object **RtDeviceConnection** registers the pointer to the **RtDeviceConnection** instance by sending a **RegisterPointer** message to the **RtRegistry** and returns the device connection object ID to object **RtDeviceConnection**. In step 1820, object **RtDeviceConnection** sends a **Create** message to object **RtDeviceProxy** to create a device proxy for the requesting device. In step 1822, object **RtDeviceProxy** creates an instance of **RtDeviceProxy** using template **RtNew**. In step 1824, object **RtDeviceProxy** sends a **GetDeviceConnectionIndex** message to the object **RtDeviceConnection** to have the index of the device in the device connection table managed by **RtDeviceConnection** for later use. In step 1826, object **RtDeviceProxy** registers the pointer to the **RtDeviceProxy** instance by sending a **RegisterPointer** message to the **RtRegistry** and returns the device proxy object ID to **RtDeviceConnection**. In step 1828, object **RtDeviceConnection** passes the sync request message to the **aRtDeviceConnection** instance for processing via the **handleInboundMessage** method. In step 1830, object **aRtDeviceConnection** sends a sync response message back to the remote device to indicate successful completion of the Device Connection creation.

Referring to **FIGURE 20**, an object communication diagram illustrates a method for sending request/ response messages between devices. The remote object communications (ROC) service in one device sends a request message to the ROC service in another device. The request message is processed and a response message is sent back to the originating device.

Prior to sending messages, a UDP device connection is established between devices. Following the sending of request/ response messages between devices, a response message from a remote device has been received and is ready for processing by ROC services.

In step 1910, a read attribute request is issued by an application program to an **aRtDeviceProxy** instance associated with a remote device. In step 1912, the **aRtDeviceProxy** instance builds a request message to be sent to the remote device to read the attribute value and asks the **RtRocReqRespService** to send the message using the **SendRequest** method. In step 1914, object **RtRocReqRespService** sends the

message to the instance of **RtDeviceConnection** associated with the connection to the remote device using the **send_msg** method. In step 1916, the instance of **RtDeviceConnection** then transmits the message to the remote device over the device connection. In step 1918, the instance of **RtDeviceConnection** in the remote device receives the message and requests the **RtRocRouter** class to route the message to the correct inbound message service. In step 1920, object **RtRocRouter** determines that the message is a request/response message and requests object **RtRocReqRespService** to **ProcessInboundReqResp**. After the message is processed by the ROC services and the message consumer a response message is built, in step 1922 object **RtRocRqtRespService** sends the response message to the originating device using the **SendResponse** method. In step 1924, the outbound message queue processing of **RtRocReqRespService** sends the response message to the instance of **RtDeviceConnection** associated with the connection to the source device using the **send_msg** method. In step 1926, the instance of **RtDeviceConnection** then transmits the response message back to the original device. In step 1928, the instance of **RtDeviceConnection** in the original device receives the message and requests the **RtRocRouter** class to route the message to the correct inbound message service. In step 1930, object **RtRocRouter** determines that the message is a request/response message and requests **RtRocReqRespService** to **ProcessInboundReqResp**.

Referring to **FIGURE 21** an object communication diagram illustrates a method downloading a network configuration. A user, following completion of the device configuration for a system, initiates a download to a controller/ multiplexer. A device table configuration script is built by the configuration application. Using communications services, the configuration application establishes a device connection with the controller/ multiplexer to receive the download and sends a download script to the controller device. The controller/ multiplexer receives the download script messages and processes the device table. In step 2010, a configuration download application program builds remote object communications (ROC) script download messages containing the device table download script. In step 2012, the Download application issues a **GetDevice** message to **RtDevice** to get the Object ID for the **RtDeviceProxy** for the remote device. In step 2014, the **RtDeviceProxy** does not yet exist so a **Create** message is sent to **RtDeviceProxyC** to create the necessary device proxy object. In step 2016, **RtDeviceProxyC** sends a **GetDeviceConnIndex** message to **RtDeviceConnection** to get the index of the device connection for the remote device in the device connection table. In step 2018, the device connection does not yet exist so a **RtDeviceConnection** object is created to manage the connection to the remote device. A lookup is performed in the database to find the remote device entry. The device communications data (for example, ID and IP Addresses) is retrieved from the database and a new entry is added to the configuration devices connection table. This connection is marked permanent in the connection table since the device initiated the connection. In step 2020, a **startActiveConnection** message is sent to the **aRtDeviceConnection** object to establish a connection to the remote device. In step 2022, the **aRtDeviceConnection** sends an **RtSyncMessage** to the remote device. In step 2024, the remote device receives the **RtSyncMessage** and attempts to find an entry in the device connection table for the sending device. In step 2026, no entry is found so a new entry is added to the device connection table for the sending device and a **RtDeviceConnection** object is created to handle the connection

in the receiving device. In step 2028, a **RtSyncReplyMessage** is created and sent back to the sending device containing the device connection index from the device table. The device connection is now established and ready to send and receive messages. In step 2030, the **RtDeviceProxyC** sends a create **RtDeviceProxyS** message to the remote device. In step 2032, the **RtDeviceProxyS** is created in the remote device. In step 2034, the Download Application sends the download scripts to the remote device via **RtRocReqRespServices** using the **SendMsg** call. In step 2036, **RtCommScriptDownload** receives the Device Table script and processes each device table item and stores the data in a database Registry used to hold configuration data. For controller/ multiplexers this processing is used to create **RtDeviceConnection** objects and add the objects to the device connection table, allowing the memory to be acquired on download rather than subsequently.

The digital control system program 115 includes a diagnostic program and display routine for viewing diagnostic information relating to a process in a coherent manner, whether the diagnostic information is derived from a Fieldbus device or a nonFieldbus device. The digital control system program 115 advantageously incorporates diagnostic information relating to all devices within the process control environment 100 and presents this information to a user in a uniform manner using the diagnostic display routine so that the control strategy and the diagnostic information are presented to a user as if all control actions and diagnostic information were performed or generated at a single location.

Referring to **FIGURE 22**, a pictorial view of a front-of-screen display illustrates a flowchart of the operations of a diagnostic display routine 2200 including a communication diagnostics program 2210. A user graphically views the status and integrity of workstations, controllers and network links within the process control environment 100 using easily-recognized icons. The diagnostic display routine 2200 also generates summary status information for a device or network link segment. The diagnostic display routine 2200 displays detailed internal integrity information concerning devices connected to the network, including packets sent and received, the number of connections and the like. Some diagnostic information is accessed via modem to implement remote diagnostics functionality.

A workstation 102 or 104 and controller/ multiplexer 110 function in combination to supply detailed diagnostic information about the status of the communications subsystem in connected devices including detailed integrity information about the status of the ACN communications links, device connections, ethernet statistics, and communications stack diagnostic information. A diagnostic check of communication functionality is also supported. The communication diagnostics 2210 support three levels of diagnostics information including basic diagnostics for the entire network, diagnostic information for a single device and diagnostic information for a single device connection.

At the basic network diagnostic level, the communication diagnostics 2210 gather information from all network devices and present the information to a user. The communication diagnostics 2210 collect information from remote controller/multiplexers 110 on demand or as a background task executing

periodically. The communication diagnostics 2210 send network communications status requests to a diagnostic port of a particular device to determine integrity of the communications path to that device.

A response to a network communications status request contains communications integrity information including device type information indicative of whether the device is a controller or workstation, and primary and secondary status summary information. The Connection Status Summary, which is set to "Bad" if an error exists on the link, is added as an attribute to a Communications Subsystem Module, allowing a user to display these values on the operator interface upon request.

The communication diagnostics 2210 request diagnostic information from each device connected into the process control environment 100 so that the status of the communications links between the device and a remote device are displayed.

The communications diagnostics 2210 monitor and display diagnostic information for a single device to supply detailed diagnostic information about communications for a particular device, including link status information for device connections and the ethernet statistics for ethernet interfaces in the device. The communications diagnostics 2210 request the device communications status using an appropriate diagnostic remote object communications (ROC) message. The device communications status includes information such as a Device Table Index, a Device ID, a Device Connection State (Ready, Synching, ACK Pending, Closed), a Primary Device Address, a Secondary Device Address, a Link Status and diagnostic information for a single device connection.

Ethernet statistical information is held in counters which start counting when the device is booted and are not reset until reboot of the device. The counts are implemented as attributes to the Communications subsystem module as part of the module attributes and include input error information including numbers of total input errors, input CRC errors and input overrun errors. Output error information includes numbers of total output errors, output late collisions, output collisions exceeding a preselected number, output overrun errors and output carrier lost errors. Other counts include the numbers of packets received, packets sent, bytes received, bytes sent, broadcast packets received, broadcast packets sent, unknown protocol messages received, transmit deferrals, single collision transmits and multiple collision transmits.

The communications diagnostics 2210 also monitor and display detailed diagnostic information for a particular device connection in a particular device. When a user requests device connection statistics, the communications diagnostics 2210 designate a destination device ID for a connection. Device connection statistics contain link status information and statistical information for the designated device connection, including a Device Connection State (Ready, Synching, ACK Pending, Closed), a Remote Primary Address (Primary ACN IP Address for the destination device), a Remote Secondary Address (Secondary ACN IP Address for this device or 0), a Primary Link Status (Good/Bad), a Secondary Link Status (Good/Bad). The statistics also include counts of messages received on Primary Link, messages received on Secondary Link,

messages sent on Primary Link, messages sent on Secondary Link, Synchs Sent, Synchs Received, ACK Time-outs on Primary Link, and ACK Time-outs on Secondary Link.

The communications diagnostics 2210 support checking of a communications path between two devices in a network. A first device sends a message, such as an evoking message, over a specified communication link to a remote device. The remote device echoes the message back to the sender. A successful evoking message and echo are indicative that the communications subsystem is functional in the remote device. This interaction is different from a normal TCP/IP ping which is handled completely by an ethernet hardware driver.

Referring to **FIGURE 23**, an object communication diagram illustrates a method for one device to check whether another device exists on a network. A diagnostic application program which is either part of the process control environment or a stand-alone application monitors to determine whether a remote device exists on an ACN network and, if so, whether the remote device is capable of communication. The diagnostic application program sends a message to the remote device and expects to receive the message echoed from the remote device or a time-out.

In step 2310, a diagnostic application program sends a Ping message to object **RtCommDiag** requesting that a communication inquiry (ping) be performed to a specified device name or address. In step 2312, object **RtCommDiag** creates a communications socket to perform the inquiry. In step 2314, object **RtCommDiag** creates an **RtEchoMessage** to send to the remote device. In step 2316, object **RtCommDiag** sends the **RtEchoMessage** to the specified device using **RtCommSendTo** and waits for the message to be echoed back from the remote device. In step 2318, object **RtDeviceConnection** in the remote device receives the **RtEchoMessage** and processes the message using the **HandleDiagnosticMessage** method. In step 2320, object **RtDeviceConnection** immediately returns an **RtEchoMessage** with a diag code of echo response to object **RtCommDiag** in the source device using **RtCommSentToMessage**. In step 2322, object **RtCommDiag** receives the echo response message and returns a successful status to the diagnostic application. If no response is received from the remote device, in step 2324, object **RtCommDiag** returns a failed status to the diagnostic application. In step 2326, object **RtCommDiag** closes the socket created to perform the inquiry.

Referring to **FIGURE 24**, an object communication diagram illustrates a method for requesting device communications diagnostics. A diagnostic application program requests, accesses and displays the status of all device connections in a remote device. The diagnostic application program sends a request for the required diagnostic information to the remote device and waits for a response. Once the response is received, all device connections and device status in the remote device are displayed to the user. This information may be communicated in multiple messages so that the diagnostic application program makes multiple requests for diagnostic data.

In step 2410, a Diagnostic Application program sends a **GetDeviceCommDiags** request to object **RtCommDiag** to request communications diagnostics for the device connections in the remote device. In step 2412, object **RtCommDiag** sends a **GetDeviceConnectionIndex** message to object **RtDeviceConnection** to request the device connection table index for the remote device. In step 2414, object **RtCommDiag** issues a **Create** to object **RtRocMessage** to create a message to be used to request diagnostic information. In step 2416, object **RtRocMessage** creates a new instance of **aRtRocMessage** to be used for the diagnostic request. In step 2418, object **RtCommDiag** builds the message to be sent and issues a **SendRequest** to **RtRocReqRespService** to send the message to the remote device. In step 2420, object **RtCommDiag** issues a **waitForResponse** to the **aRtRocMessage** instance created for the diagnostic request message. In step 2422, object **RtRocReqRespService** performs a **send_msg** on the instance of **aRtDeviceConnection** associated with the remote device. In step 2424, the **aRtDeviceConnection** instance transmits the request message to the remote device. In step 2426, object **RtDeviceConnection** in the remote device receives the message and issues a **handleInboundMessage** to the **aRtDeviceConnection** instance associated with the sending device. In step 2428, the **aRtDeviceConnection** instance passes the message off to **RtRocRouter** via the **Route** method for processing. In step 2430, object **RtRocRouter** responds to the message by issuing a **ProcessInboundReqResp** to the **RtRocReqRespService**. In step 2432, the **RtRocReqRespService** determines the message type from the request message ID and issues a **perform** command on the **RtRocDevConDiagListMsg** message. In step 2434, **RtRocDevConDiagListMsg** sends a **GetDiagList** message to object **RtDeviceConnection** to get the device connection diagnostic data for the device connections in this device. In step 2436, object **RtDeviceConnection** puts the diagnostic data into a data buffer using various fill routines provided by the utility class **RtDevConDiagListData**. In step 2438, **RtRocDevConDiagListMsg** then performs a **CreateResponse** to create the response message containing the diagnostic data. In step 2440, **RtRocDevConDiagListMsg** issues a **storeOn** message to **RtDevConDiagListData** to put the diagnostic data into the response message. In step 2442, **RtRocDevConDiagListMsg** sends the response message back to the requesting device through **RtRocReqRespService** using the **SendResponse** method. In step 2444, **RtRocReqRespService** performs a **send_msg** to the **aRtDeviceConnection** instance associated with the requesting device. In step 2446, the **aRtDeviceConnection** instance transmits the response message back to the requesting device. In step 2448, the **aRtDeviceConnection** instance in the requesting device receives the response message from **RtDeviceConnection** via a **handleInboundMessage**. The response message is then sent to **RtRocRouter** using the **Route** method. In step 2450, **RtRocRouter** performs a **ProcessInboundReqResp** on **RtRocReqRespService**. In step 2452, **RtRocReqRespService** informs the **aRtRocMessage** instance that the response to the original request for diagnostic data is available. In step 2454, **RtCommDiag** issues a **readFrom** to **RtDevConDiagListData** to retrieve the diagnostic data from the response message. In step 2456, **RtCommDiag** passes the data back to the Diagnostic Application which issues **getData** messages to **RtDevConDiagListData** to retrieve the diagnostic data for display. The requesting device returns a buffer containing the general diagnostics for all device connections actively in place.

Referring to **FIGURE 25**, an object communication diagram illustrates a method for requesting device connection communications diagnostics. A diagnostic application program requests, accesses and displays the detailed status of a single device connection existing in a remote device. The diagnostic application program sends a request for device connection statistics information to the remote device and waits for a response. Once the response is received, the device connection statistics for the device connection are displayed to the user. In step 2510, a Diagnostic Application program sends a **GetDeviceConDiags** request to **RtCommDiag** to get the device connection statistics for a specific device connection in the remote device. In step 2512, **RtCommDiag** sends a **GetDeviceConnectionIndex** message to **RtDeviceConnection** to get the device connection table index for the remote device. In step 2514, **RtCommDiag** issues a **Create** to **RtRocMessage** to create a message to be used to request the diagnostic information. In step 2516, **RtRocMessage** creates a new instance of **aRtRocMessage** to be used for the diagnostic request. In step 2518, **RtCommDiag** builds the message to be sent and issues a **SendRequest** to **RtRocReqRespService** to send the message to the remote device. In step 2520, **RtCommDiag** issues a **waitForResponse** to the **aRtRocMessage** instance created for the diagnostic request message. In step 2522, **RtRocReqRespService** performs a **send_msg** on the instance of **aRtDeviceConnection** associated with the remote device. In step 2524, the **aRtDeviceConnection** instance transmits the request message to the remote device. In step 2526, **RtDeviceConnection** in the remote device receives the message and issues a **handleInboundMessage** to the **aRtDeviceConnection** instance associated with the sending device. In step 2528, the **aRtDeviceConnection** instance passes the message to **RtRocRouter** via the **Route** method for processing. In step 2530, **RtRocRouter** determines that the message is a request response type message and issues a **ProcessInboundReqResp** to the **RtRocReqRespService**. In step 2532, the **RtRocReqRespService** determines the message type from the request message ID and issues a **perform** command on the **RtRocDevConDiagDetailMsg** message. In step 2534, **RtRocDevConDiagDetailMsg** sends a **GetDiagDetail** message to **RtDeviceConnection** to get the device connection statistics for the requested device connection. In step 2536, **RtDeviceConnection** puts the diagnostic data into a data buffer using various fill routines provided by the utility class **RtDevConDiagDetailData**. In step 2538, **RtRocDevConDiagDetailMsg** performs a **CreateResponse** to create the response message containing the diagnostic data. In step 2540, **RtRocDevConDiagDetailMsg** issues a **storeOn** message to **RtDevConDiagDetailData** to put the diagnostic data into the response message. In step 2542, **RtRocDevConDiagDetailMsg** sends the response message back to the requesting device through **RtRocReqRespService** using the **SendResponse** method. In step 2544, **RtRocReqRespService** performs a **send_msg** to the **aRtDeviceConnection** instance associated with the requesting device. In step 2546, the **aRtDeviceConnection** instance transmits the response message back to the requesting device. In step 2548, the **aRtDeviceConnection** instance in the requesting device receives the response message from **RtDeviceConnection** via a **handleInboundMessage**. The response message is sent to **RtRocRouter** using the **Route** method. In step 2550, **RtRocRouter** performs a **ProcessInboundReqResp** on **RtRocReqRespService**. In step 2552, **RtRocReqRespService** informs the **aRtRocMessage** instance that the response to the original request for diagnostic data is available. In step 2554, **RtCommDiag** issues a

readFrom to RtDevConDiagDetailData to retrieve the diagnostic data from the response message. In step 2556, RtCommDiag passes the data back to the Diagnostic Application which issues getData messages to RtDevConDiagDetailData to retrieve the diagnostic data for display.

5 Generally, a controller/ multiplexer 110 is manually added to a network. For example, a device is typically added to a network when a user selects the ACN Segment onto which the device is to be connected and issues a 'New Device' command. The user is prompted for the device type, either workstation or controller/ multiplexer, the device name and a comment field. A configuration system automatically assigns the next Device ID to the device and builds a default name based on the Device ID. Optionally, the configuration system automatically generates primary and secondary IP addresses and subnet masks for the device based on PCN number and the previously assigned Device ID.

Alternatively, a controller/ multiplexer is automatically sensed and incorporated into a run-time system as shown in FIGURE 26. In step 2610, a controller/ multiplexer, upon connection to the ACN and application of power, automatically sends a request for identification or verify IP address assignment. The request message includes the MAC address of the controller/ multiplexer. The request is handled by a "Plug&Play Network Configuration Service", which is known in the operating system art, at a master configuration controller/ multiplexer in step 2612. In step 2614, the "Plug&Play Network Configuration Service" receives the request from the network to assign/ verify an IP address, searches a table of configured devices for a MAC address match. If a match is found, in step 2616 the "Plug&Play Network Configuration Service" responds with the Device Name, Device ID, IP Address Information, Subnet Mask Information, ACN Segment Number and other items included in the Device Table. If no match is found, in step 2618 the "Plug&Play Network Configuration Service" automatically generates a default name for the device based on the controller/ multiplexer MAC address (for example, Controller-000001). The new device is added to the database in a Device Scratch area in step 2620.

In step 2622, using the Explorer™ a user selects each unassigned controller/ multiplexer in the Device Scratch area, drags the selection to the appropriate ACN segment and, and either adds the selection to the system as a new device or drops the selection to a pre-existing device configuration. If the unassigned controller/ multiplexer is added as a new device, the configuration processing proceeds in the manner of manual incorporation of the device. In step 2624, a user is prompted for the real device name using the previously assigned name 'Controller-000001' as a default. If automatic address assignment is set, the new device is assigned the next Device ID and associated IP addresses and Subnet masks are automatically assigned in step 2626. If manual address assignment is set, the device is automatically assigned the next Device ID and the user is prompted to enter the IP Addresses and Subnet Masks in step 2628. The MAC address for the controller/ multiplexer is set to the MAC address of the 'Controller-000001' as dragged into the ACN segment. The new controller/ multiplexer Name, Device ID, IP Address, Subnet Masks and ACN number are added to the device table in the database. The next request by an unconfigured controller/ multiplexer is answered by the "Plug&Play Network Configuration Service".

If a new controller/ multiplexer is dragged and dropped over an existing device, that device must be a controller/ multiplexer type device and have an unassigned MAC address. Accordingly, the MAC address of the previously entered controller/ multiplexer is set to the MAC address of the 'Controller-000001' device which was dropped. The new controller/ multiplexer Name, Device ID, IP Addresses, Subnet Masks and
5 ACN number are available for sending to the requesting controller/ multiplexer by the "Plug&Play Network Configuration Service".

The digital control system program 115 includes an auto-configure routine for automatically configuring the input/output (I/O) subsystem in response either to an "auto-configure" command by a user or in response to detection of a new controller/multiplexer.

10 Referring to **FIGURE 27**, a flow chart illustrates steps of an automatic configuration routine for configuring a physical I/O device. An auto-configure command may be directed to a Controller/Multiplexer 110, causing each I/O subsystem in the Controller/Multiplexer 110 to auto-configure. An auto-configure command may be directed to an I/O subsystem, causing each I/O Card in the I/O subsystem to auto-configure. An auto-configure command may also be directed to an I/O Card.

15 The auto-configure operation for an I/O Card first interrogates the I/O Card at a particular card position to determine a Card Type in step 2710 and, implicitly for some I/O Cards, the number of I/O Ports in the I/O Card. If no I/O Card is previously created in the engineering database for that card position, an I/O Card of the appropriate type is defined and the appropriate number of I/O Ports are created in step 2712. If an I/O Card does exist in the engineering database for that card position, but the Card Type in the
20 engineering database does not match the Card Type sensed at the card position, the auto-configure operation generates a graphic notification of the mismatch in step 2714 and interrogates a user to determine whether the engineering database is to be changed to include the sensed Card Type. The Card Type in the engineering database is changed to the sensed Card Type in step 2716 if requested by the user.

25 Once the Card Type is known, the auto-configuration program interrogates each I/O Port in accordance with the Card Type in step 2718 to determine the Port Type and, if information is available, the number of I/O Devices on the I/O Port. If no I/O Port is previously created in the engineering database for that port address, an I/O Port of the appropriate type is defined and the appropriate number of I/O Devices are created in step 2720. If an I/O Port exists in the engineering database for the Port address, but the Port Type does not match the type of the sensed I/O Port, the user is notified of the mismatch in step 2722, and
30 asked whether the engineering database is to be changed to match the sensed I/O Port in step 2724. The Port Type in the engineering database is changed to the sensed Port Type in step 2726 if requested by the user.

35 Once the Port Type is known, the auto-configuration program interrogates each I/O Device in accordance with the Port Type in step 2728 to determine the Device Type. If no I/O Device is previously created in the engineering database for that device address, an I/O Device of the appropriate type is defined in step 2730. If an I/O Device exists in the engineering database for the Device address, but the Device

Type does not match the type of the sensed I/O Device, the user is notified of the mismatch in step 2732, and asked whether the engineering database is to be changed to match the sensed I/O Device in step 2734. The Device Type in the engineering database is changed to the sensed Device Type in step 2736 if requested by the user.

5 In step 2738, instrument signal tags (ISTs) are automatically created for primary signal sources on the I/O Ports and I/O Devices, unless an IST already exists with the identical signal source path.

Referring to **FIGURE 28**, a front-of-screen display, also called a "screen" 2800, for a graphical user interface (GUI) depicts a display of a system configuration. The screen 2800 depicts navigation selections which are operated by a user to select, construct and operate a process control configuration. The navigation
10 program supplies an initial state for navigating across various tools and processors in a network. A user controls the navigation program to access libraries, areas, process control equipment and security operations.

The illustrative system configuration is described and controlled with respect to a control system setup 2802, control strategies 2804, and a physical setup 2806. The functions of automatically sensing and automatically configuring a control system relate to the physical setup 2806. In particular, the functions of
15 automatically sensing and automatically configuring physical devices in a control system relate to the commission and activation of devices in the control network 2808, and the decommissioning of controllers 2810.

In an illustrative embodiment, a process control system controls various devices attached to a process control network in accordance with a Fieldbus standard protocol. In the Fieldbus protocol, a standard
20 user application is defined based on blocks. A block is a representation of various different types of application functions. Types of blocks include resource blocks, function blocks, and transducer blocks.

A resource block describes characteristics of a fieldbus device such as a device name, manufacturer, and serial number. A device includes only a single resource block.

A function block defines the control system behavior. Input and output parameters of function
25 blocks may be linked over the fieldbus. The execution of each function block is precisely scheduled. A user application may include numerous function blocks. Examples of standard function blocks include analog input (AI), analog output (AO), bias (B), Control Selector (CS), Discrete Input (DI), Discrete Output (DO), Manual Loader (ML), Proportional/ Derivative (PD), Proportional/ Integral/ Derivative (PID) and ratio (RA). Function blocks are built into fieldbus devices to define a selected device functionality. In one example, a
30 simple temperature transmitter may contain an AI function block. A control valve often includes a PID function block and an AO block.

A transducer block decouples function blocks from local input and output functions for reading sensors and commanding output hardware. Transducer blocks contain information such as calibration data

and sensor type. Typically a user application uses one transducer block for each input or output function block.

Another object defined in the user application includes link objects for defining the links between function block inputs and outputs internal to the device and across the fieldbus network. Trend objects allow local trending of function block parameters for access by other devices. Alert objects are used to allow reporting of alarms and events on the fieldbus. View objects are predefined groupings of block parameter sets that are used in the human/ machine interface. The function block specification defines four views for each type of block.

Referring to **FIGURE 29**, a state transition diagram illustrates the various states of a field device.

The field device states include an offline state 2902, an unrecognized state 2904, a standby state 2906, a commissioned state 2908, and an unbound state 2910. The state of a field device is determined by several parameters including a system management state (SM-State), a physical device tag (PD-Tag), a device address, device revision information (Rev*), and a device identification (Device-ID). In the illustrative embodiment, a device in the commissioned state 2908 is a Fieldbus device that is available for control strategy configuration and installation. A decommissioned device is a device that has been removed from the commissioned state 2908.

Several events occur that generate a state transition of a plurality of state transitions T1 through T14. One or more actions are performed during each state transition.

A state transition T1 is caused by the event in which a field device residing at a temporary address is queried with a system management identify service (SM-IDENTIFY) and the query determines that the device has a cleared physical device tag. The state transition T1 changes from a NULL state to an OFFLINE state by allocating a standby address for the field device. Executing logic, typically in the form of firmware, software, or hardware, executes a set physical device tag service (SET-PD-TAG) to set the physical device tag identical to the device identification of the field device. Executing logic also uses a set device address service (SET-ADDRESS) to send a standby address to the field device.

A state transition T2 is caused by the event in which a field device residing at a temporary address is queried with a system management identify service (SM-IDENTIFY) and the query determines that the device has a physical device tag that is identical to the device identification of the device. The state transition T2 changes from a NULL state to an OFFLINE state by allocating a standby address for the field device. Executing logic uses a set device address service (SET-ADDRESS) to send a standby address to the field device.

A state transition T3 is caused by the event in which a field device residing at a temporary address is queried with a system management identify service (SM-IDENTIFY) and the query determines that the device has a physical device tag and a device identification configured for the current process control system

network link. The state transition T3 changes from a NULL state to an OFFLINE state using executing logic that employs the set device address service (SET-ADDRESS) to send an assigned address to the field device.

5 A state transition T4 is caused by the event in which a field device residing at a temporary address is queried with a system management identify service (SM-IDENTIFY) and the query determines that the device has a physical device tag and a device identification not configured for the current process control system network link. The state transition T4 changes from a NULL state to an UNRECOGNIZED state.

10 A state transition T5 is caused by an event in which the device appears at a temporary address and the device is being commissioned by a user. The state transition T5 changes from an OFFLINE state to an OFFLINE state using executing logic, typically in the form of firmware, software, or hardware, that executes a set physical device tag service (SET-PD-TAG) to clear the physical device tag of the field device. Executing logic also executes a set physical device tag service (SET-PD-TAG) to send an assigned physical device tag to the field device. Executing logic further uses a set device address service (SET-ADDRESS) to send an assigned address to the field device.

15 A state transition T6 is caused by an event in which the device appears at a temporary address and the device is being decommissioned by a user. The state transition T6 changes from an OFFLINE state to an OFFLINE state using executing logic that executes a set physical device tag service (SET-PD-TAG) to clear the physical device tag of the field device.

20 A state transition T7 is caused by an event in which a user requests to place a decommissioned device in standby. The state transition T7 changes from an OFFLINE state to an OFFLINE state by allocating a standby address for the field device. Executing logic executes a set physical device tag service (SET-PD-TAG) to set the physical device tag identical to the device identification of the field device. Executing logic also uses a set device address service (SET-ADDRESS) to send a standby address to the field device.

25 A state transition T8 is caused by an event in which the field device appears at the standby address. The state transition T8 changes from an OFFLINE state to a STANDBY state through executing logic that reads device revision information from the resource block.

A state transition T9 is caused by an event in which the field device appears at the assigned address. The state transition T9 changes from an OFFLINE state to a COMMISSIONED.

30 A state transition T10 is caused by a user requesting to commission a device in the STANDBY state. The state transition T10 changes from the STANDBY state to the OFFLINE state through executing logic that uses a clear address service (CLEAR-ADDRESS) to clear the device address.

A state transition T11 is caused by a user requesting to decommission a device in the STANDBY state. The state transition T11 changes from the STANDBY state to the OFFLINE state through executing logic that uses a clear address service (CLEAR-ADDRESS) to clear the device address.

A state transition T12 is caused by a user requesting to decommission a device in the COMMISSIONED state. The state transition T12 changes from the COMMISSIONED state to the OFFLINE state through executing logic that uses a clear address service (CLEAR-ADDRESS) to clear the device address.

5 A state transition T13 is caused by a user requesting to decommission a device in the INITIALIZED state of the Fieldbus system management states. The state transition T13 changes from the UNRECOGNIZED state to the OFFLINE state through executing logic that executes a set physical device tag service (SET-PD-TAG) to clear the physical device tag of the field device.

10 A state transition T14 is caused by a user requesting to decommission a device in the SM-OPERATIONAL state of the Fieldbus system management states. The state transition T14 changes from the UNRECOGNIZED state to the OFFLINE state through executing logic that uses a clear address service (CLEAR-ADDRESS) to clear the device address.

15 In accordance with the Fieldbus standard, to operate properly a Fieldbus device has a unique device address (network address) and a unique physical device tag. Each device connected to the process control system network link has a unique node designator. A data link specification specifies a range of allowable values for node designators including a range for permanent devices, a range for temporary addresses, and a range for visitor devices. The temporary addresses are used by devices that are not presently in the SM-OPERATIONAL state. The Fieldbus interface maintains partitioning of the address space for permanent devices into three sets. One set, called "assigned addresses", includes addresses assigned to devices with a specific physical device tag, regardless of whether the device is present on the bus. The assigned addresses is assigned using a software engineering tool on the basis of information input by a user relating to Link-Active-Scheduler takeover preference. A second set, termed "standby addresses", describes devices in the SM-OPERATIONAL state but have no device addresses assigned. The standby addresses are managed by the Fieldbus interface. The third set of addresses are addresses outside the first and second sets and refer to unused addresses.

25 The small number of temporary addresses complicates autosensing and online address assignment. Standby addresses are defined and utilized to support functionality of the autosensing and online address assignment operations. The assigned address set and the standby address set are defined to be equal to the number of potential devices connected to the process control system network link. For example, if sixteen devices may be potentially connected to the process control system network, then sixteen assigned addresses are defined and sixteen standby addresses are defined.

30 The device revision information includes a manufacturer's identification (MANUFAC-ID), a device type (DEV-TYPE), a device revision (DEV-REV), and a device description revision (DD-REV).

In the offline state 2902 a field device is recently attached to a process control system network or is in the process of being commissioned or decommissioned. The offline state 2902 includes device states having a plurality of parameter combinations. In a first offline state 2902, the system management state is UNINITIALIZED and the physical device tag is cleared. In a second offline state 2902, the system management state is INITIALIZED and the physical device tag is read from the physical device and displayable on a screen. In either of the offline states 2902, the device address is a temporary address, the revision information is not available, and the device identification is read from the device and displayable on the screen.

In the unrecognized state 2904, the field device physical device tag and the device identification do not match the values that are commissioned for a device that is connected to the process control system network. The unrecognized state 2904 includes device states having a plurality of parameter combinations. In a first unrecognized state 2904, the system management state is INITIALIZED with a device address that is a temporary address. In a second unrecognized state 2904, the system management state is SM-OPERATIONAL with a device address that is a standby address or an assigned address. In either unrecognized state 2904, the physical device tag is read from the device and displayable on the screen, the device revision is not available, and the device identification is read from the device and displayable on the screen.

In the standby state 2906, the field device is not yet autosensed and is therefore not available for configuration in the control strategy or included in Link-Active-Scheduler (LAS) schedules of the system management configuration. In the standby state 2906, function block execution and link communications are disabled. Note that a Link-Active-Scheduler is a deterministic centralized bus scheduler that includes a list of transmit times for all data buffers in all devices that are to be cyclically transmitted. When a device is due to send a data buffer, the Link-Active-Scheduler issues a compel data (CD) message to the device. Upon receipt of the CD message, the device broadcasts or "publishes" the data in the buffer to all devices on a field device bus and the broadcasting device is defined to be a "publisher". Any device that is configured to receive the data is defined to be a "subscriber". Scheduled data transfers are typically used for the regular, cyclic transfer of control loop data between devices on the fieldbus.

In the standby state 2906, the system management state is SM-OPERATIONAL, the physical device tag is equal to the device identification, and the device address is a standby address. The device revision information is read from the field device and displayable. The device identification is read from the device and displayable on the screen.

The unbound state 2910 is a configuration placeholder for a field device that is to be physically attached subsequently. The unbound state 2910 supports configuration of control strategies utilizing the function blocks in a field device that is not yet connected. In the unbound state 2910, the system management state is not yet applicable but the physical device tag is specified by a user and the device

address is assigned by the user. The device revision information set according to the most recent commission or configuration. The device identification is cleared.

In the commissioned state 2908, the field device is available for control strategy configuration and installation. The system management state is SM-OPERATIONAL, the physical device tag is specified by a user, and the device address is assigned by the user. The device revision information is read from the field device and displayable on the screen. The device identification is read from the field device, stored in a field configuration database, and displayable on a display screen.

Several operations or "use cases" are defined for controlling commissioning and decommissioning of field devices.

Referring to **FIGURE 30**, a flow chart illustrates a first operation or "use case" which describes an operation of commissioning a new device 3000. Prior to the commissioning of the new device, the Fieldbus interface is operational. A device is connected to the process control system network. The device either has no physical device tag or has a physical device tag that is equal to the device identification.

The operation of commissioning a new device 3000 results in a condition in which the device is assigned a new physical device tag and a device address, and the device is ready for function block configuration. The new field device is entered into the process control system network database with the device identification bound and the device revision information set. An engineering software tool that displays the process control system network status displays the new device as a COMMISSIONED device.

In a first step 3002, the field device appears in the "live list" at a temporary address. A "live list" is a list of all devices that are properly responding to a pass token (PT) message. All devices on a fieldbus are allowed to send unscheduled messages between the transmission of scheduled messages. The Link-Active-Scheduler grants permission to a device to use the fieldbus by issuing a pass token (PT) message to the device. When the device receives the PT, it is allowed to send messages until the messages are complete or until a maximum allotted token hold time has expired. As a highest priority activity, the Link-Active-Scheduler accesses a CD schedule containing a list of actions that are set to occur on a cyclic basis. At a scheduled time, the Link-Active-Scheduler sends a compel data (CD) message to a specific data buffer in the fieldbus device. The device immediately broadcasts a message to all devices on the fieldbus. The Link-Active-Scheduler performs remaining operations between scheduled transfers. The Link-Active-Scheduler continually adds new devices to the field bus by periodically sending probe node (PN) messages to addresses that are not on the live list. If a device is present at the address and receives the PN, the device immediately returns a probe response (PR) message. If a device responds with the PR message, the Link-Active-Scheduler adds the device to the live list and confirms by sending the device a node activation (NA) message. A device remains on the live list so long as the device responds properly to PTs. When a device is added or removed from the live list, the Link-Active-Scheduler broadcasts changes to the live list to all devices to allow each device to maintain a current copy of the live list.

In a second step 3004, the interface queries the field device using a system management identify service (SM-IDENTIFY) and determines whether the field device is in the UNINITIALIZED state with no physical device tag set or in the INITIALIZED state having a physical device tag that is equal to the device identification. The interface then allocates 3006 a standby address for the field device.

5 A logical step 3008 directs that a previously UNINITIALIZED device, in step 3010, sets the physical device tag of the field device identical to the device identification using a set physical device tag service (SET-PD-TAG), thereby placing the field device in the INITIALIZED state. The standby address is sent to the field device 3012 using a set address service (SET-ADDRESS), advancing the field device from the INITIALIZED state to the SM-OPERATIONAL state. At this point the field device appears in the "live list" 10 at a standby address 3014. Device revision information is read from the resource block 3016. In step 3018, an executing software engineering tool displays the field device as a STANDBY device.

In step 3020, a new user assigns a new physical device tag to the field device. The physical device tag is constrained to be unique and not the same as the device identification. During the assignment of the physical device tag, a device address is assigned to the field device using a software engineering tool and the 15 Link-Active-Scheduler takeover preference is set to "selectable". The device revision information is read from the field device and written to the process control system network database. The interface changes the state of the field device 3022 to the INITIALIZED state using a clear address service (CLEAR-ADDRESS). The field device appears in the "live list" at a temporary address 3024.

In a step 3026, the interface queries the field device using a system management identify service 20 (SM-IDENTIFY) and recognizes the field device by the device identification. The interface uses the set physical device tag service (SET-PD-TAG) to clear the physical device tag 3028, thereby changing the field device state to the UNINITIALIZED state. The set physical device tag service (SET-PD-TAG) is then used to send the assigned physical device tag to the field device 3030, changing the field device state to the INITIALIZED state. The set address service (SET-ADDRESS) is called to send the assigned address to the 25 field device 3032, placing the field device in the system management operational state (SM-OPERATIONAL). The field device appears in the "live list" at the assigned address 3034. In the process control system network database, the device identification is bound 3036 to the device. The software engineering tool displays the field device as a COMMISSIONED device.

Referring to FIGURE 31, a flow chart illustrates a second operation or "use case" which describes 30 an operation of commissioning an unbound device 3100. Prior to the commissioning of the unbound device, the Fieldbus interface is operational. A field device has been created in the process control system network database and a physical device tag and a device address are assigned to the field device. However, the field device is not bound to a device identification. The process control system network database has also been initialized to contain device revision information read from the field device. A software engineering tool 35 displays the field device as an UNBOUND device. The UNBOUND device to be commissioned is either a

field device with no physical device tag or a field device having a physical device tag that is identical to the device identification. The UNBOUND field device is commissioned to place the field device on the process control system network link.

5 The operation of commissioning an UNBOUND device 3100 results in a condition in which the device is configured with a physical device tag and an assigned device address, and the device is ready for function block configuration. The new field device is entered into the process control system network database with the device identification bound. An engineering software tool that displays the process control system network status displays the device as a COMMISSIONED device.

10 In a first step 3102, the field device appears in the "live list" at a temporary address. In a second step 3104, the interface queries the field device using a system management identify service (SM-IDENTIFY) and determines whether the field device is in the UNINITIALIZED state with no physical device tag set or in the INITIALIZED state having a physical device tag that is equal to the device identification. The interface then allocates 3106 a standby address for the field device.

15 A logical step 3108 directs that a previously UNINITIALIZED device, in step 3110, sets the physical device tag of the field device identical to the device identification using a set physical device tag service (SET-PD-TAG), thereby placing the field device in the INITIALIZED state. The standby address is sent to the field device 3112 using a set address service (SET-ADDRESS), advancing the field device from the INITIALIZED state to the SM-OPERATIONAL state. At this point the field device appears in the "live list" at a standby address 3114. Device revision information is read from the resource block 3116. In step 3118,
20 an executing software engineering tool displays the field device as a STANDBY device.

In step 3120, a user assigns a physical device tag to the field device by associating the field device with the pre-configured device. The device revision information is read from the field device to ascertain that the information matches the device revision information in the process control system network database for the pre-configured device. If the device revision information of the device does not match the database, the
25 user may override the database, reading the device revision information from the field device and writing the information to the process control system network database. Alternatively, the device revision information for an UNBOUND device may be made blank, allowing any physical device to be bound with the UNBOUND device. The interface changes the state of the field device 3122 to the INITIALIZED state using a clear address service (CLEAR-ADDRESS). The field device appears in the "live list" at a temporary address 3124.

30 In a step 3126, the interface queries the field device using a system management identify service (SM-IDENTIFY) and recognizes the field device by the device identification. The interface uses the set physical device tag service (SET-PD-TAG) to clear the physical device tag 3128, thereby changing the field device state to the UNINITIALIZED state. The set physical device tag service (SET-PD-TAG) is then used to send the assigned physical device tag to the field device 3130, changing the field device state to the
35 INITIALIZED state. The set address service (SET-ADDRESS) is called to send the assigned address to the

field device 3132, placing the field device in the system management operational state (SM-OPERATIONAL). The field device appears in the "live list" at the assigned address 3134. In the process control system network database, the device identification is bound 3136 to the device. The software engineering tool displays the field device as a COMMISSIONED device.

5 Referring to FIGURE 32, a flow chart illustrates a third operation or "use case" which describes an operation of decommissioning a device 3200. A field device is decommissioned for several reasons. For example, when a Fieldbus device is obsolete, a user may wish to clear a network interconnection structure of nonfunctioning branches so that the process control system no longer expends resources on the obsolete device. Also, a user may suspect that a Fieldbus device is malfunctioning and degrading operations of a
10 segment of a network interconnection structure. The user may diagnose the problem by having the process control system ignore the suspected Fieldbus device temporarily to determine whether the remaining devices in the segment operate properly.

Prior to the operation of decommissioning a device, the Fieldbus interface and the field device are operational and the field device appears in the live list at the assigned or standby address. A software
15 engineering tool displays the field device as a COMMISSIONED or STANDBY device. The software engineering tool executes a routine that prepares the field device for decommissioning, for example by clearing function block tags and clearing an OPERATIONAL-POWERUP flag.

The operation of decommissioning a device results in a condition in which the physical device tag of the field device is cleared and the field device is prepared to be removed from the process control system
20 network link. The process control system network database entry for the field device designates the device identification as in an unbound condition. The software engineering tool displays the device identification as an UNBOUND device and displays the physical device as an OFFLINE device.

The operation of decommissioning a device 3200 begins when a user selects a "Decommission" operation for the field device 3202. A graphic user interface includes a software engineering tool that issues
25 a "Decommission" command to an appropriate controller within the process control system. The decommission command specifies a target I/O subsystem, card and port identifiers, and the device identification of the field device to be decommissioned. The device identification is specified since another device with the same physical device tag may be present in an UNRECOGNIZED state. The interface changes the state of the field device 3204 to the INITIALIZED state using a clear address service (CLEAR-
30 ADDRESS). The field device appears in the "live list" at a temporary address 3206.

In a step 3208, the interface queries the field device using a system management identify service (SM-IDENTIFY) and recognizes the field device by the physical device tag and the device identification. The interface uses the set physical device tag service (SET-PD-TAG) to clear the physical device tag 3210, thereby changing the field device state to the UNINITIALIZED state.

In the process control system network database, the device identification is unbound and the software engineering tool displays the field device as an UNBOUND device 3212. In a next step 3214, the software engineering tool displays the field device as an OFFLINE device.

5 A network interface card stores a designation that the field device has been decommissioned 3216 and does not move the field device to a STANDBY address unless directed by the user. If the decommissioned device is not move to a STANDBY address, the interface card tracks the field device until the field device advances off the live list.

Referring to **FIGURE 33**, a flow chart illustrates a fourth operation or "use case" which describes an operation of attaching a commissioned device without enablement of operational powerup 3300. Prior to
10 the operation of attaching a commissioned device 3300, the Fieldbus interface is operational. The configuration of the Fieldbus interface includes the field device in an attached condition. The physical device tag and the device identification of the field device are matched. Following the operation of attaching a commissioned device 3300, the field device has an assigned address.

The field device appears in the "live list" at a temporary address 3302. In a step 3304, the interface
15 queries the field device using a system management identify service (SM-IDENTIFY) and recognizes the field device by the physical device tag and the device identification as part of the Fieldbus interface configuration. The set address service (SET-ADDRESS) is called to send the assigned address to the field device 3306, placing the field device in the system management operational state (SM-OPERATIONAL). The field device appears in the "live list" at the assigned address 3308.

20 Referring to **FIGURE 34**, a flow chart illustrates a fifth operation or "use case" which describes an operation of replacing a device 3400. A device is replaced by decommissioning the current field device 3402 connected to the process control system network link, if possible, and commissioning a replacement to the UNBOUND device 3404. The step of decommissioning the current field device 3402 is described in detail with reference to **FIGURE 32**. The step of commissioning a replacement to the UNBOUND device 3404 is
25 described with reference to **FIGURE 31**.

Referring to **FIGURE 35**, a flow chart illustrates a sixth operation or "use case" which describes an operation of attaching an UNRECOGNIZED device 3500. Prior to the operation of attaching a commissioned device 3300, the Fieldbus interface is operational. A field device is attached which has a physical device tag and a device identification that is not configured for the current process control system
30 network link. Following the operation of attaching an UNRECOGNIZED device 3500, the field device is identified and the software engineering tool displays the device as a UNRECOGNIZED device. The operation of attaching an UNRECOGNIZED device 3500 may be performed without use of the software engineering tool.

The field device appears in the "live list" 3502. In a step 3504, the interface queries the field device using a system management identify service (SM-IDENTIFY) and determines that the physical device tag and the device identification do not match a field device in the present configuration.

5 Referring to FIGURE 36, a flow chart illustrates a seventh operation or "use case" which describes an operation of decommissioning an unrecognized device 3600. Prior to the operation of decommissioning an unrecognized device, the Fieldbus interface is operational. The field device is identified which has a physical device tag and a device identification that are not configured for the present process control system network link. A software engineering tool displays the field device as an UNRECOGNIZED device.

10 The operation of decommissioning an unrecognized device 3600 results in a condition in which the physical device tag of the field device is cleared and the field device is prepared to be removed from the process control system network link. The software engineering tool displays the field device as an OFFLINE device.

The operation of decommissioning an unrecognized device 3600 begins when a user selects a "Decommission" operation for the field device 3602. A graphic user interface includes a software
15 engineering tool that issues a "Decommission" command to an appropriate controller within the process control system. The decommission command specifies a target I/O subsystem, card and port identifiers, and the device identification of the field device to be decommissioned.

If the field device is in the INITIALIZED state, logic step 3604 directs the decommissioning operation 3600 to a clear the physical device tag step 3612. Otherwise, the interface changes the state of the
20 field device 3606 to the INITIALIZED state using a clear address service (CLEAR-ADDRESS). The field device appears in the "live list" at a temporary address 3608.

In a step 3610, the interface queries the field device using a system management identify service (SM-IDENTIFY) and recognizes the field device by the physical device tag and the device identification. The interface uses the set physical device tag service (SET-PD-TAG) to clear the physical device tag 3612,
25 thereby changing the field device state to the UNINITIALIZED state. In a next step 3614, the software engineering tool displays the field device as an OFFLINE device.

A network interface card stores a designation that the field device has been decommissioned 3616 and does not move the field device to a STANDBY address unless directed by the user. If the decommissioned device is not move to a STANDBY address, the interface card tracks the field device until
30 the field device advances off the live list.

Referring to FIGURE 37, a flow chart illustrates an eighth operation or "use case" which describes an operation of placing a decommissioned device in a standby condition 3700. Prior to the operation of

placing a decommissioned device in a standby condition 3700, the Fieldbus interface is operational. A field device is decommissioned and in the OFFLINE state.

5 The operation of placing a decommissioned device in standby 3700 results in a condition in which the field device is placed at a standby address with the physical device tag of the field device set identical to the device identification. The software engineering tool displays the field device as a STANDBY device.

The operation of placing a decommissioned device in standby 3700 begins when a user selects a "Place in Standby" operation for the field device 3702. A graphic user interface includes a software engineering tool that issues a "Place in Standby" command to an appropriate controller within the process control system 3704. The decommission command specifies a target I/O subsystem, card and port identifiers, and the device identification of the field device to be placed in standby.

10

The interface allocates a standby address 3706 for the field device. The set physical device tag service (SET-PD-TAG) is then used to set the physical device tag identical to the device identification 3708, changing the field device state to the INITIALIZED state. The set address service (SET-ADDRESS) is called to send the standby address to the field device 3710, placing the field device in the system management operational state (SM-OPERATIONAL). The field device appears in the "live list" at the standby address 3712. Device revision information is read from the resource block 3714. In step 3716, an executing software engineering tool displays the field device as a STANDBY device.

15

A user may subsequently commission the field device 3718, either by creating a new device or binding the field device to an UNBOUND device in the process control system network database. The techniques for commissioning a device are described with respect to FIGUREs 30 and 31.

20

Referring to FIGURE 38, a schematic block diagram illustrates a program structure of a process control configuration program for defining a process configuration using a plurality of control languages. The module editor 3820 is a software program that executes on the CPU 116 within a workstation such as the engineering workstation 106. Using the module editor 3820, a user activates one language editor of multiple control language editors including an attribute editor program 3830, a Function Block Edit/View/Debug program 3840, a Sequential Function Chart program 3850, a Ladder Logic program 3870 and a Structured Text program 3890. The multiple control language editors operate with compatible databases and interfaces so that a common control strategy is defined using the different control language editors. The look and feel of the different control language editors is similar so that a user easily and efficiently may use different editors for different purposes to best take advantage of different aspects of the languages.

25

30

The module editor 3820 is the fundamental routine for configuration entry and is used to create, edit, and re-use control and equipment module instances and library modules defined by the SP88 standard. The module editor 3820 supplies a graphical technique for configuring, visualizing, and navigating multiple

modules that combine to form a control strategy. The module editor 3820 is also used for module linking and defining module containment.

The module editor 3820 supplies access to all aspects of configuration of a module including support for algorithm definition, as well as information gathering relating to alarms, events, history, condition, help, components and attributes.

The module editor 3820 includes various features that facilitate navigation through a control system. For example, the module editor 3820 filters attributes on user-defined key words to supply a focused attribute configuration view. Furthermore, the module editor 3820 includes a fast install capability which defaults to performing the install to all devices affected by changes to the module. The control strategy is transferred from the editors into runtime engines using an install script. A plurality of control language execution engines execute the control strategy on one or more controller/ multiplexers 110.

Off-line configuration supports 'work in progress' for an existing module's control strategy. When the new strategy is complete, it can be installed to the existing module. This 'work in progress' can be flagged to not allow it to be installed.

The attribute editor program 3830 is a control program operated by a user to view and edit parameter values associated with a control module or control algorithm elements including function blocks, ladders, composites, and the like. The attribute editor program 3830 is used on-line to configure a control module. On-line, the attribute editor program 3830 is used to view and edit data. Attributes are defined to furnish module-level access to parameters contained within a module. The attribute editor program 3830 is activated by the module editor 3820 and other control language editors. The attribute editor program 3830 presents a simple attribute configuration view using a graphical navigation tree pathway display to consolidate the parameters to be configured for a module. A user activates the attribute editor program 3830 and navigates through the graphical navigation tree to find a particular attribute. When an attribute is selected, the attribute is displayed either using an application window or as a dialog depending on the context from which the display is evoked. The attribute editor program 3830 facilitates configuration by supporting copy, paste and bulk change of attributes using a drag and drop technique. The attribute editor program 3830 is used to define attribute parameter values but not to install attributes since attributes are installed when the module containing the attribute is installed. As attribute parameter values are entered by a user, the attribute editor program 3830 performs error checking. The attribute editor program 3830 allows a user to sort and view data with typical spreadsheet capabilities, including expansion and reduction of the number of columns, or by containment.

The Function Block Edit/View/Debug program 3840 is a control program operated by a user to define a control strategy by assembling function block elements designated under the Fieldbus Foundation and IEC 1131-3 standards. The function block program 3840 is implemented in a function block execution engine which operates in a controller/ multiplexer 110. The function block execution engine executes

function block control algorithms using a Function Block Library 3842. The function block program 3840 supplies a graphical editor for creating, editing and reusing control strategies using function blocks stored within a Function Block Library 3802 and other defined control modules, for example control modules defined for usage by other application programs. A created control strategy is saved as a reusable library component in the Function Block Library 3802 for subsequent usage or for usage by another application program. The function block program 3840 also creates, edits and reuses control structures for HART and FF timing field devices. The function block and module constituents of a control strategy are not confined to a particular device, but are rather partitioned across multiple control devices, including HART and FF timing field devices.

A function block or module includes attributes that are modified using the function block program 3840. Any attribute in the system, unless security protected, is accessed through the navigation tree pathway for that attribute and is read or written from the function block editor.

The function block editor program 3840 supports an off-line creation and debug function for control strategies of modules and reusable library components. During off-line operation, a control device is not connected to the network. Offline operation is useful for initializing attributes and parameters before the control device is connected using an attribute editor 3842 of the function block editor program 3840 since, during off-line operation, multiple control strategies can be open, and configurations can be copied and pasted between strategies.

The function block editor program 3840 also supports on-line graphical editing, viewing and debugging of functions blocks during operation, including display of block input and output values. Selective on-line edit and debug mode options include single step, forced inputs and breakpoint options, for example. One window is used for on-line editing and debugging so that the user may correct an inconsistent parameter value in place. Input values during on-line editing are checked for consistency with off-line editing values, and disallowed if inconsistent.

Hardcopy output information generated by the function block editor program 3840 includes graphical representation and configuration information such as block attribute and parameter values.

Several features of the function block program 3840 facilitate user support including the creation of reusable library components, user conversation support, and input error checking. User conversation support within step actions allows a user to configure pop-up dialogs with responses. Reusable library components are developed to simplify engineering during configuration and execution so that a single Function Block Diagram (FBD) acts as a subroutine for multiple modules or FBDs during execution.

The Sequential Function Chart (SFC) Edit/View/Debug program 3850 is a control program operated by a user to define a control strategy by assembling steps, step actions and transitions designated under IEC 1131-3 standards. The sequential function chart program 3850 is implemented in an SFC execution engine

which operates in a controller/ multiplexer 110. The SFC execution engine executes sequential function chart control algorithms including steps, actions and transitions. The sequential function chart program 3850 supplies a graphical editor for creating, editing and reusing sequential control strategies using sequential function charts. The sequential function chart program 3850 supports execution of parallel logic paths. Executing sequential function charts are associated with a module, at least for the duration of SFC execution.

A sequential function chart includes attributes that are modified using the sequential function chart program 3850. Any attribute in the system, unless security protected, is accessed from a step in a sequential function chart execution.

The sequential function chart program 3850 supports an off-line creation and debug function for control strategies of modules. During off-line operation, a control device is not connected to the network. Offline operation is useful for initializing attributes and parameters before the control device is connected using an attribute editor 3842 of the sequential function chart program 3850 since, during off-line operation, multiple control strategies can be open, and configurations can be copied and pasted between strategies.

The sequential function chart editor program 3850 also supports on-line graphical editing, viewing and debugging of sequential function charts during operation, including a "live" display of attribute values. Selective on-line edit and debug mode options include single step, forced inputs and breakpoint options, for example. On-line editing includes setting of a breakpoint for executing a control strategy while changes are made in the strategy. One window is used for on-line editing and debugging so that the user may correct an inconsistent parameter value in place. Input values during on-line editing are checked for consistency with off-line editing values, and disallowed if inconsistent. The sequential function chart program 3850 supports runtime view including a display of steps, actions, currently-executing transactions and attribute values.

Hardcopy output information generated by the sequential function chart program 3850 includes graphical representation and configuration information such as step actions.

Several features of the sequential function chart program 3850 facilitate user support including the creation of reusable library components, user conversation support, and input error checking. User conversation support within step actions allows a user to configure pop-up dialogs with responses. Reusable library components are developed to simplify engineering during configuration and execution so that a single Sequential Function Chart (SFC) acts as a subroutine for multiple modules or SFCs during execution.

The Ladder Logic program 3870 is a control program operated by a user to define a control strategy using ladder elements and function blocks in combination in a ladder logic diagram. The ladder logic program 3870 is implemented in a Ladder Logic engine which operates in a controller/ multiplexer 110. The ladder logic engine executes ladder control algorithms including coils, contacts, and several standard function blocks. The Ladder Logic program 3870 includes a basic discrete control ladder logic library 3872

which includes the components for solving basic discrete control operations. These operations are logical extensions of IEC 1131-3 standards including elements such as coils, contacts, timers, flip/flops, edge triggers and counters with one output connection defined per ladder rung. The Ladder Logic program 3870 also supports placement of user-defined blocks within a ladder logic diagram. A ladder logic diagram is applicable to a function block that supports power flow in which the state of execution is determined by whether "power" is flowing through a "rung" in the ladder. Power flow is somewhat analogous to data flow in a logic diagram. The applicable function blocks, alone, are displayed by the Ladder Logic program 3870 on a ladder logic palette.

The Ladder Logic program 3870 may be exclusively used by a user to configure and debug a control strategy or a ladder diagram sheet may be used as a composite within another ladder logic sheet, a function block diagram sheet or an sequential function chart sheet. A ladder logic sheet can hold composite ladder logic diagrams, function block diagrams or sequential function chart sheets. A ladder logic element can read and write to attributes in other sheets and in other modules.

Ladder logic diagrams support both on-line and off-line editing. User can switch from debug functionality to the edit mode on a specific ladder logic diagram to make structural changes in the control strategy. The user may change any parameter from either the debug view or the edit view.

The Ladder Logic program 3870 includes a debug functionality that includes a display of power flow, parameter values and energized or de-energized states. The ladder logic debug functionality is the substantially the same as the debug functionality of the Sequential Function Chart editor program 3850 in which the user forces input values, sets breakpoints and the like. The Ladder Logic program 3870 supports ladder logic simulation, historical data collection and mode, alarm and status report generation.

Referring to FIGURE 39A, a screen presentation of a configuration screen 3900 is depicted which is generated by a configuration program using the function block editor 3840. The configuration screen 3900 includes a navigation portion 3902 and a screen-specific portion 3904. The navigation portion 3902 includes navigation tabs 3910 which allow a user to access particular sections of the configuration program. In the illustrative state of the configuration screen 3900, the configuration program is awaiting a user entry of a configuration command. Navigation tabs 3910 indicate several primitive functions for the entry into a function block including a navigation tab 3912 for selecting a simple set dominant (SR) flip-flop, a navigation tab 3914 for selecting a simple subtract block, a navigation tab 3916 for selecting a simple timed pulse block, a navigation tab 3918 for selecting a simple transfer block, and a navigation tab 3920 for selecting a simple exclusive-OR (XOR) block. The navigation tabs 3910 also include a block assistant 3922 tab for inserting a function block. The screen-specific portion 3904 illustrates a function block 3924 that is to be configured, including configuration of attributes and connections with other function blocks.

The block assistant 3922 tab is selected to insert a function block.

Referring to **FIGURE 39B**, a Selection screen 3930 is displayed in response to the selection of an insert function block. The selection screen 3930 also includes a selection portion 3932 and a screen-specific portion 3934. The screen-specific portion 3934 includes an entry block 3936 for entering the name of a new block that is to be created. The selection portion 3932 includes a plurality of selection tabs 3938 including a function block 3940 tab, an embedded block 3942 tab, a linked composite 3944 tab and a module block 3946 tab. The selection portion 3932 also includes a variety of buttons which furnish navigation functions, including a Back button 3948, a Next button 3950 and a Cancel button 3952. The Back button 3948 takes the user to the previous screen presentation in strict historical order. The Next button 3950 takes the user to the screen presentation appropriate for the selections that are made on the current screen presentation. The cancel button 3952 terminates the selection.

The embedded block 3942 tab is selected to insert an embedded block.

Referring to **FIGURE 39C**, a Choice screen 3960 is evoked to allow the user to select a control language editor for configuring the embedded block, either a function block editor using selection 3962 or a sequential function chart editor using selection 3964.

The selection 3964 is chosen to utilize the sequential function chart editor.

Referring to **FIGURE 39D**, a screen presentation of a configuration screen 3970 is illustrated in which the function block 3924 is configured using the function editor 3840 and a newly created block 3972 is configured using the sequential function chart editor 3850.

Editing of newly created block 3972 is selected so that the sequential function chart editor 3850 is invoked.

Referring to **FIGURE 39E**, details of a the newly created block 3972 are shown in the screen-specific portion 3934 of the configuration screen 3980 including attributes 3982 and a step 3984. The navigation portion 3986 of the configuration screen 3970 no longer lists navigation tabs relating to function blocks, but rather includes navigation tabs 3988 relating to a step 3990, a transition 3992, a termination 3994, and input terminal 3996 and an output terminal 3998 to define the steps and transitions of a sequential function.

Referring to **FIGURE 40**, an object model shows object relationships of various objects for handling alarm and event functions. Various conditions are defined to be "events" including Alarms, Alarm acknowledgments, user changes (writing attributes, invoking methods, log-in/out), configuration changes to the "run-time" system (installations, dc-installs, etc.), Sequential Function Chart (SFC) state changes, Operator Attention Requests (OARs), and other miscellaneous Events (non-alarm state transitions including equipment state changes).

A common characteristic for all types of events is that the occurrence or state transition of an event can be recorded in a Event Journal. All events are associated with one (or more) plant areas. Event occurrence records (RtEventOccurrenceRecord 4040) are captured in the Event Journal, or Journals, (RtEventJournal 4020) designated for the associated plant area (RtPlantArea 4010).

5 A user activates the Event Journal, typically using a workstation, by configuring one or more Plant Areas within which the activated Event Journal captures events. On-line operation of the Event Journal is modified under user control by disabling or enabling specified classes of events to be recorded.

The user configures an Alarm behavior by creating Alarm Attributes (RtAttribute 4032) in Control Modules or Equipment Modules (RtModule 4030). An Alarm Attribute furnishes reference to any boolean
10 Attribute within the Control Module or Equipment Module containing the Attribute. Alarm Attributes are created only at the Module level. Alarm Attributes are not created in Composite Function Blocks.

Referring to **FIGURE 41**, a state transition diagram illustrates alarm attribute states. A user either disables or enables an Alarm Attribute. When disabled 4110, the Alarm Attribute appears in a "normal" condition, called an "Inactive and Acknowledged" condition. The Disabled/Enabled condition of an Alarm
15 Attribute is changed either on-line, by an Operator, or automatically by a control algorithm in the system. The initial Disabled/Enabled condition is set at configuration time. An enabled Alarm Attribute has either an Active condition 4116 or 4118 or an Inactive condition 4112 or 4114. The Alarm is Active ("in alarm") when the referenced boolean Attribute is TRUE. The Alarm Attribute is optionally configured to invert the sense of the alarm state, so an Alarm Attribute with .INV characteristic TRUE operates is if the referenced
20 Attribute's value of FALSE indicates an "in alarm" condition. The Active/Inactive condition is driven by the state of the referenced Attribute so that the Active/Inactive condition is not directly changed by the Operator or another control algorithm.

While Enabled, an Alarm Attribute has either an Acknowledged 4112 or 4116 or Unacknowledged state 4114 or 4118. The Alarm Attribute is placed in the Unacknowledged condition only if the Alarm
25 Attribute makes a transition from Inactive to Active state, unless automatically acknowledged. An Operator or another control algorithm may acknowledge the Alarm, changing the Alarm Attribute to the Acknowledged condition.

An Alarm Attribute is either automatically acknowledged (AACK'd) or not automatically acknowledged. AACK is determined from the current Alarm priority. A user-configured, system-wide table
30 determines AACK behavior. For example, the table may designate that all "LOW" priority Alarms are automatically acknowledged (AACK is TRUE), all "MEDIUM" and "HIGH" priority Alarms are not automatically acknowledged (AACK is FALSE). When AACK is TRUE, the alarm is never placed in an Unacknowledged condition.

The combined operation of the Enable/Disabled, Active/Inactive, and Acknowledged/Unacknowledged conditions results in user-visible states for an Alarm Attribute that are shown in FIGURE 41.

An enabled Alarm Attribute initially goes to the Inactive/Ack'd State 4112, and may immediately transition to either Active/Unack'd 4114 or Active/Ack'd 4116. The transition to Active/Ack'd 4116 is accompanied by a standard "transition" behavior for Alarms in which the transition is timestamped and the event is recorded in the event journal, for example.

The Alarm Attribute has multiple fields that provide a user-visible interface. A CUALM "current alarm" field is "OK" when the Alarm Attribute is in the Disabled state 4110, the Inactive/Ack'd state 4112 or Inactive/Unack'd state 4114. Otherwise CUALM is the word/value associated with the configured Alarm Type. A DESC "description" field has an alarm description that is generated when the Alarm Attribute changes state. The DESC field is initialized to the empty string. A LAALM "latched alarm" field is "OK" when the Alarm Attribute is in the Disabled state 4110 or the Inactive/Ack'd state 4112. Otherwise the LAALM field is the word/value associated with the configured Alarm Type. The LAALM field presents "latched" alarm activations, enabling Acknowledgment even if the duration of being Active is very short. A NALM "unacknowledged alarm" field indicates the Acknowledged/Unacknowledged condition of the Alarm Attribute. The NALM field is used to determine when alarm summary entries can be removed. An ENAB "alarm enabled" field indicates the current Enabled condition for the Alarm Attribute. An IENAB "alarm initially enabled" field indicates the configured Enabled condition for the Alarm Attribute. An INV "inverted input" field indicates whether the value of an associated boolean Attribute is inverted before alarm processing. The INV configurable characteristic permits an Alarm Attribute reference normally TRUE boolean Attributes, for example an Attribute holding a discrete input. A PRI "alarm priority" field indicates current priority (High/Medium/Low) of an Alarm Attribute. TABLE II shows the boolean Attributes as follows:

| TABLE II. Attribute: (user defined boolean Attribute) | | | | |
|---|--|---|--|--------------------------|
| Field Name | Access/Privilege | A (ASCII) | F (Float) | B (Binary) |
| CV (or none) | (same as LAALM) | (same as LAALM) | (same as LAALM) | (same as LAALM) |
| CUALM | Read: [view] Write: N/A Config: N/A | "OK" (alarm word) | 0 (alarm type number ¹) | 0 (alarm type number) |
| DESC | Read: [view] Write: N/A Config: N/A | (description generated at the time of alarm state change) | N/A | N/A |
| ENAB | Read: [view] Write: [alarms] Config: N/A (init to IENAB) | "NO" "YES" | 0.0 1.0 | 0 1 |
| INV | Read: [view] Write: N/A | "NO" "YES" | 0.0 1.0 | 0 1 |

| | | | | |
|-------|--|--------------------------------------|--------------------------|------------------------|
| | Config: [config] | | | |
| IENAB | Read: [view] Write: N/A Config: [config] Instance overrideable | "DISABLE" "ENABLE" | 0.0 1.0 | 0 1 |
| LAALM | Read: [view] Write: N/A Config: N/A | "OK" (alarm word) | (alarm type number) | (alarm type number) |
| NALM | Read: [view] Write: [operate] Config: N/A (init FALSE) | "NO" "YES" | 0.0 1.0 | 0 1 |
| PRI | Read: [view] Write: [alarms] Config: [config] Instance overrideable | "LOG" "LOW" "MEDIUM" "HIGH" | 3.0 2.0 1.0 0.0 | 3 2 1 0 |

The process control system 100 consolidates many potential Active Alarm conditions into a short list of "highest priority" alarms. A selection criteria is used to select the highest priority alarms for consolidation. The selection criteria includes analysis, in order of decreasing preference, the Acknowledged condition with the Unacknowledged condition having precedence over the Acknowledged condition, the Alarm Priority in order of High then Medium then Low precedence, and the Time of Detection with the newest timestamp gaining precedence.

Alarm Consolidation is available at the SP88 Module level and the Plant Area level. Alarm consolidation is accessed via a pre-defined Attribute name "ALARMS" available on SP88 Modules (Control & Equipment) and Plant Areas. ALARMS is an indexed Attribute, where the index selects the Nth highest priority alarm in the consolidation (e.g. ALARMS[1] accesses the highest priority Alarm, ALARMS[2] accesses the second highest priority Alarm, etc.) Thus, for example, 'AREA1/FIC101/ALARMS[1]' references the highest priority Alarm in Module FIC101 and 'AREA1/ALARMS[2]' references the second highest priority Alarm in Plant Area AREA1. The maximum index supported on the ALARMS Attribute is 5.

The Fields supported on the ALARMS[N] Attribute include the LAALM "latched alarm" field which indicates the Active Or Unacknowledged conditions of the Nth priority Alarm Attribute. The LAALM Field is in the alarm word, or alarm type value, of the Nth priority Alarm Attribute during the ACTIVE/UNACK'D, ACTIVE/ACK'D, or INACTIVE/UNACK'D states. The NALM "unacknowledged alarm" field indicates the Acknowledged/Unacknowledged condition of the Nth priority Alarm Attribute. The PRI "alarm priority" field indicates the configured priority (High/Medium/Low) of the Nth priority Alarm Attribute. A TAG "alarm Tag" field returns a fully qualified Attribute reference string (excluding Field) for the Nth priority Alarm Attribute. A MODULE "alarm Module" field indicates the SP88 Module (tag) which has the Nth priority Alarm. The MODULE tag returns a Module reference string which can be used to access the "primary control display" attribute for that Module.

Some Fields are supported on the ALARMS Attribute only at the SP88 Module level. If an index value is applied for these fields, the index value is ignored. The SP88 Module level ALARMS Attribute fields include an ENAB "alarm enabled" field which indicates the current Alarm Enabled condition for the Module. A PRIAD "priority adjustment" field is a number (normally 0) which is added to the current

- 5 Priority of each Alarm Attribute in the Module when determining the effective Priority of an Alarm. The PRIAD Module-wide adjustment is used to decrease the Alarm Priorities of all the alarms in a Module, permitting diminished Alarm visibility. TABLE III describes the ALARMS Attribute, in detail, as follows:

| TABLE III. Attribute: ALARMS[N] (N = 1..5, 1 is highest priority alarm) | | | | |
|---|---|--|---------------------------------------|---------------------------------------|
| Field Name: | Access/Privilege | AZ (ASCII) | FE (Decimal) | B (Binary) |
| CV (or none) | (same as LAALM) | (same as LAALM) | (same as LAALM) | (same as LAALM) |
| ENAB | Read: [view] Write: [alarm] Config: N/A (init TRUE) | "NO" "YES" | 0.0 1.0 | 0 1 |
| LAALM | Read: [view] Write: N/A Config: N/A | "OK" (alarm word of Nth alarm) | 0 (alarm type number of Nth alarm) | 0 (alarm type number of Nth alarm) |
| MODULE | Read: [view] Write: N/A Config: N/A | Module name for Nth alarm attribute. | N/A | N/A |
| NALM | Read: [view] Write: N/A Config: N/A | "NO" "YES" | 0.0 1.0 | 0 1 |
| PRI | Read: [view] Write: N/A Config: N/A | "LOW" "MEDIUM" "HIGH" (priority of Nth alarm) | 2.0 1.0 0.0 | 2 1 0 |
| PRIAD | Read: [view] Write: [alarm] Config: N/A (init 0) | "0".."3" | 0.0..3.0 | 0..3 |
| TAG | Read: [view] Write: N/A Config: N/A | Full reference path for Nth alarm attribute. | N/A | N/A |

- User-Level Alarm Consolidation is achieved using an "Alarm Banner" in the graphical user interface. Alarm consolidation is supported for a "current user". Each user is granted authority over one or more Plant Areas. The "current user" alarm consolidation provides the ability to present the highest priority alarms of the set of Plant Areas currently in the user's span of control. A pre-defined "attribute container" exists, named "THISUSER", which supports the ALARMS[N] Attribute. Users optionally construct displays referencing "THISUSER/ALARMS[N].field" to allow quick access to the highest priority alarms for the "current user".

- 15 A user Enables and Disables Alarms if granted an alarm privilege. With the alarm privilege, a user may Enable or Disable a single Alarm Attribute by writing to the .ENAB field (e.g. writing TRUE to 'AREA1/FIC101/HIALM.ENAB'. Each Alarm Attribute in the process control system includes a single Enable/Disable condition. When one user changes state, Alarms are Enabled/Disabled for all users. With

the alarm privilege, a user may Enable or Disable all alarms in a SP88 Module Alarm by writing to the .ENAB field of the ALARMS Attribute (e.g. writing TRUE to 'AREA1/FIC101/ALARMS.ENAB'). Disabling Alarms at the Module level overrides, but does not overwrite, individual Alarm's .ENAB condition. When Alarms are Enabled at the Module level, Alarm processing determined by the individual Alarm's .ENAB condition is restored.

Each SP88 Module in the system includes a single Enable/Disable condition. When one user changes state, all Alarms in that Module are Enabled/Disabled for all users.

A user having an operate privilege can Acknowledge Alarms. With the operate privilege, a user Acknowledges a single Alarm Attribute by writing FALSE to the .NALM field (e.g. writing FALSE to 'AREA1/FIC101/HIALM.NALM'). Attempts to write TRUE to the .NALM are ignored. Each Alarm Attribute in the process control system has a single Acknowledge condition. When one user changes state, Alarms are Acknowledged for all users. With the operate privilege, a user may Acknowledge all alarms in a SP88 Module Alarm by writing FALSE to the .NALM field of the ALARMS Attribute (e.g. writing FALSE to 'AREA1/FIC101/ALARMS.NALM'), an operation which has substantially the same effect as writing FALSE to the .NALM field of all Alarm Attributes in the Module. Attempts to write TRUE to the .NALM are ignored.

A user having an alarm privilege can change alarm priority. With the alarm privilege, a user may change PRI on a single Alarm Attribute by writing to the .PRI field (e.g. writing 0 to 'AREA1/FIC101/HIALM.PRI' to make it a "HIGH" priority alarm). Since Auto Acknowledgment behavior is determined by Alarm Priority, changing Alarm Priority may cause an Alarm to change acknowledgment status. For example, changing from a priority with AutoAck FALSE to a priority with AutoAck TRUE should cause unacknowledged alarms to be acknowledge. Also, changing from a priority with AutoAck TRUE to a priority with AutoAck FALSE should cause acknowledged alarms to become unacknowledged.

Each Alarm Attribute in the system has a single .PRI condition. When one user changes state, .PRI is changed for all users.

With the alarm privilege, a user may adjust the effective priority for all alarms in a SP88 Module Alarm by writing to the .PRIAD field of the ALARMS Attribute. For example, a user writing 1 to 'AREA1/FIC101/ALARMS.PRIAD' increases the current Alarm Priority value, thereby diminishing the annunciation behavior, by one "step" so that HIGH priority becomes MEDIUM priority, and LOW priority becomes LOG. The user only sets PRIAD to positive numbers and therefore is only used to diminish normal annunciation behavior. Setting PRIAD to 0 reestablishes the "normal" priorities determined per Alarm Attribute. Effective alarm priorities are not adjusted "below" LOG. Since Auto Acknowledgment behavior is determined by Alarm Priority, changing PRIAD at the Module level may cause individual Alarms to change acknowledgment status.

Each SP88 Module in the system has a single .PRIAD value. When one user changes state, all Alarms in that Module are affected for all users.

Alarms are viewed using a display under a standard FIX™ Alarm Summary. The FIX™ graphic and display program, which is marketed by Intellution of Norwood, MA, is well known in the computing arts. The FIX™ Alarm Summary Link is the primary method to view filtered and sorted lists of Active Alarms. All capabilities of the Alarm Summary Link are supported, with the following exceptions or extensions. First, a "Tagname" column shows the process control system Attribute reference path, excluding the Field name, for the Alarm Attribute (e.g. 'AREA1/FIC101/HIALRM'). Second, a "Description" contains a user-configured string that is constructed at the time the Alarm is detected so that an Alarm captures the value of up to two Attributes at the point the Alarm was first detected. Only one "Alarm" per "Tag" is possible so that multiple Alarms may be shown for each SP88 Module in the Alarm Summary. Third, a "Time Last" entry contains the time of the last state transition for the Active Alarm, which could be the time of acknowledgement, or the time of the last transition between Active/Inactive for an unacknowledged alarm. Fourth, a "Node" column entry, which shows FIX SCADA Node source for the Alarm, is meaningless for process control system Alarms so that a by Area Filter and Sort feature are lost. Fifth, all Alarms are mapped into one of the FIX Alarm types to achieve foreground color based on the Alarm Type.

A user can access an Alarm State Transition Journaling record. Alarm state transitions are recorded in the Event Journal assigned to a Plant Area. All alarm state transitions shown in FIGURE 41 are recorded in the Event Journal, including transitions between the Inactive/Unack'd 4114 and the Active/Unack'd state 4118. Thus an operator viewing the LAALM field in displays or alarm summaries does not see transitions between Active/Inactive states for unacknowledged alarms, these transitions are recorded in the Event Journal.

Event journal entries for alarm state transitions include: (1) a timestamp of the alarm state transition as determined by the device (e.g. controller) detecting the alarm condition, (2) an "alarm" event type which distinguishes from other event journal entries, (3) a user-defined alarm category, (4) a current alarm priority, (5) an alarm word string as configured in the system Alarm Type table, (6) a new alarm state, (7) an attribute reference string or path for the alarmed attribute, and (8) a description string assembled from the description string configured in the Alarm Type table, with the configured (up to two) Attribute values inserted in the string.

A Event Journal browser application presents data in a manner shown in TABLE IV, generally sorted by timestamp and filtered on event type = "ALARM" and attribute reference string = "FIC101/PID1/HIALM"):

TABLE IV

| | | | | | | | |
|------------------------|-----------|-------------|------------|----------|---------------|-----------------------|--------------------------|
| DA-MO-YR 10:11:04.4 | ALAR M | PROC ESS | MEDI UM | HIG H | ACT/UNA CK | FIC101/PID1/HI ALM | value 96.2 limit 95.0 |
|------------------------|-----------|-------------|------------|----------|---------------|-----------------------|--------------------------|

| | | | | | | | |
|------------------------|-----------|-------------|------------|----------|-----------------|-----------------------|--------------------------|
| DA-MO-YR 10:11:18.6 | ALAR M | PROC ESS | MEDI UM | HIG H | INACT/UN ACK | FIC101/PID1/HI ALM | value 93.7 limit 95.0 |
| DA-MO-YR 10:13:45.6 | ALAR M | PROC ESS | MEDI UM | HIG H | DISABLE D | FIC101/PID1/HI ALM | value 93.1 limit 95.0 |
| DA-MO-YR 10:22:00.1 | ALAR M | PROC ESS | MEDI UM | HIG H | ACT/UNA CK | FIC101/PID1/HI ALM | value 95.8 limit 95.0 |
| DA-MO-YR 10:22:20.9 | ALAR M | PROC ESS | MEDI UM | HIG H | ACT/ACK | FIC101/PID1/HI ALM | value 95.9 limit 95.0 |
| DA-MO-YR 10:27:59.4 | ALAR M | PROC ESS | MEDI UM | HIG H | CLEAR | FIC101/PID1/HI ALM | value 94.0 limit 95.0 |

Alarm state transitions events in the Event Journal are distinct from operator change journal entries, although operator changes cause corresponding alarm state changes. For example, as shown in TABLE V as follows:

TABLE V

| | | | | | | |
|------------------------|------------|------------|---|--|-----------------------------|----------------------|
| DA-MO-YR 10:13:45.9 | CHAN GE | CJON ES | A | | FIC101/PID1/HI ALM. ENAB | new value = FALSE |
| DA-MO-YR 10:22:00.1 | CHAN GE | CJON ES | A | | FIC101/PID1/HI ALM. ENAB | new value = TRUE |
| DA-MO-YR 10:22:21.3 | CHAN GE | CJON ES | A | | FIC101/PID1/HI ALM. NALM | ALARM/ACK |
| DA-MO-YR 15:28:59.4 | CHAN GE | BSMI TH | A | | FIC101.ENAB | new value = FALSE |

The Alarm Attributes are configured, thereby setting the Alarm behavior and presentation, using the described sequence of operations. First, an "Alarm Types" Table and an "Alarm Annunciation" Table are configured. Second, in an optional step, the user-defined alarm conditions are configured, setting the boolean Attributes. Third, Alarm Attributes are created to reference the boolean Attributes, thereby identifying the System "Alarm Type", priority, and the like. Fourth, Module "instances" are created based on Module Definitions that contain Alarms. Fifth, a presentation of Alarm information is inserted into displays (pictures) via database links, dynamic color links, and Alarm Summary links. Sixth, the "Alarm Types" and "Alarm Annunciation" Tables are configured.

The "Alarm Type" table has several functions, including (1) acting as a system (Site) wide common resource which defines a common Alarm presentation behavior to speed the Alarm configuration process for each Alarm, (2) encouraging standard alarm messaging in Summaries and History Journals to improve query and analysis that information, (3) mapping alarms into FIX Alarm States.

The "Alarm Types" Table contains columns including an Alarm Type, an Alarm Word, a category and a description string column. The Alarm Type column contains a brief description of the Alarm Type, which is used to select the appropriate Type when creating an Alarm Attribute. The Alarm Word column includes a string that is returned when reading the A_CUALM or A_LAALM Fields when the Alarm is Active. The category column describes a user defined word recorded in the Event Journal used to help

filter/sort queries. The description string appears in the Alarm Summary Link and contains up to two place holders for Attribute value substitution at Alarm Detection time.

The "Alarm Types" Table default content is shown in TABLE VI as follows:

TABLE VI

| Alarm Type | Alarm Word | Category | Description String | Alarm Status |
|-----------------------|----------------|------------|--|--------------|
| Communication Error | COMM | INSTRUMENT | Communication Error | 197 |
| Open Circuit Detected | OCD | INSTRUMENT | Open Circuit Detected | 193 |
| General I/O Failure | IOF | INSTRUMENT | General I/O Failure | 192 |
| Floating Point Error | FLT | SYSTEM | Floating Point Error | 137 |
| Over Range | OVER | INSTRUMENT | Over Range Value %P1 | 67 |
| Under Range | UNDER | INSTRUMENT | Under Range Value %P1 | 66 |
| Statistical Alarm | ERROR | SYSTEM | Statistical Alarm Type %P1 Value %P2 | 12 |
| New Alarm | NEW | SYSTEM | New Alarm Value %P1 | 14 |
| Any Alarm | ANY | SYSTEM | Any Alarm Value %P1 | 13 |
| Change From Normal | CFN | PROCESS | Change From Normal Value %P1 | 7 |
| Change of State | COS | PROCESS | Change of State | 6 |
| High High Alarm | HIHI | PROCESS | High High Alarm Value %P1 Limit %P2 | 3 |
| Low Low Alarm | LOLO | PROCESS | Low Low Alarm Value %P1 Limit %P2 | 1 |
| Rate of Change | RATE | PROCESS | Rate of Change Rate %P1 Limit %P2 | 3 |
| High Alarm | HIGH | PROCESS | High Alarm Value %P1 Limit %P2 | 3 |
| Low Alarm | LOW | PROCESS | Low Alarm Value %P1 Limit %P2 | 2 |
| Deviation Alarm | DEV | PROCESS | Deviation Alarm Target %P1 Actual %P2 | 3 |
| Normal State | OK | | Normal State | 0 |
| (user defined) | (user defined) | | (user defined) | N/A |

5 Standard alarm types match the alarms types supported in FIX™.

The "Alarm Annunciation" table is a system (Site) wide common resource which furnishes a common definition of Alarm annunciation behavior to speed the Alarm configuration process for each Alarm.

10 The "Alarm Types" Table contains the columns including an Alarm Priority, an Auto Acknowledgement, a WAV file and a PC speaker frequency column. The Alarm Priority designates the Alarm priority word (HIGH/MEDIUM/LOW/LOG). The Auto Acknowledgment column contains a YES/NO

² Probably no reason for this to be configured, or even visible to the user.

value indicating if alarms of this priority should be automatically acknowledged when detected and providing an opportunity to make less important alarms less "distracting". The WAV file contains a filename of a NT compatible .WAV file which is played (looping) when an Alarm is detected (within the scope of the current user) at a Workstation with a sound card. Omitting this file name indicates that no .WAV file should be played for alarms with this priority. The PC Speaker frequency sets a value used to play a tone on the PC speaker when an Alarm is detected, within the scope of the current user, at a Workstation without a sound card. A value of 0 indicates that the PC speaker should not be used for alarms with this priority. If a .WAV file and a non 0 speaker frequency are specified for the same alarm priority, the PC speaker is used only if no sound card is present.

The "Alarm Annunciation" Table default content is shown in TABLE VII as follows:

TABLE VII

| Alarm Priority | Automatic Ack | WAV File | PC Speaker Freq. |
|----------------|---------------|--------------|------------------|
| HIGH | NO | ALRMHIGH.WAV | ? |
| MEDIUM | NO | ALRMMED.WAV | ? |
| LOW | YES | ALRMLOW.WAV | ? |
| LOG | YES | | 0 |

A user attaches Alarm (behavior) to boolean Attributes by creating Alarm Attributes according to the SP88 Module Definition which reference another boolean Attribute in the same Module.

A user create an Alarm Attribute by entering: (1) a target Attribute by path or by drag-and-drop, for example, (2) a boolean Attribute defining the alarm condition, (3) an Alarm Type selected from the system-wide list of Alarm Types, (4) an Alarm Priority (High/Medium/Low/Log), (5) an Initial Alarm Enable condition (YES/NO), (6) an Invert Input (YES/NO), (7) an optional Name of Attribute having a value to be substituted for %P1, (8) an optional Name of Attribute having a value to be substituted for %P2.

Items 7 & 8, the Attribute names, are restricted to Attributes in the same SP88 Module and are specified as "module relative" attribute references (e.g. "SP" and "PID1/PV" rather than "FIC101/SP" or "FIC101/PID1/PV").

The user also creates Module "instances" based on Module Definitions that contain Alarms. When a Module instance is created, all Alarm behavior specified in the Module Definition applies to the Module instance. The .PRI and .ENAB fields may be overridden on Alarm'd Attributes when the Module instance is created. For example, if for an Alarm Attribute named 'HIGHLIMITED', PRI is MEDIUM when the Module Definition is constructed, then when the Module instance is created, 'HIGHLIMITED.PRI' may be overridden to be LOW for this instance.

Alarms are supported for Device/Subsystem Attributes in Controllers. Attributes are defined for devices and device subsystems to provide access to information about the operation of the control system and connections to other systems. The Attributes are accessible via diagnostic tools and via pre-defined or user-

defined displays. It is valuable for some of these Attributes, especially "consolidation" Attributes, to participate in the Alarm system to draw attention to abnormal conditions in the control system.

Users create Control Modules (instances) to implement a desired "Device Alarm" strategy for Controllers and Subsystems, including processor, communications, I/O, redundancy, and the like.

5 Using Function Block algorithms, the Device/Subsystem Attributes are accessed most efficiently in the same device, but also supported for other Controllers and Workstations. Device and Subsystem Attributes are used as inputs to Function Blocks, which then can be configured to applying alarm limits on these Attributes, converting these values to boolean Attributes. AlarmAttributes then reference the boolean Attributes. In general, the full algorithm definition capability of the Function Block system is used to build
10 simple or complex Device Alarming schemes.

A number of the pre-defined Alarm Types, which are consistent with FIX™ alarm types, are suitable for distinguishing "Device Alarm" presentation and behavior. Pre-defined Control Module Definitions are supplied providing fairly comprehensive Device Alarm "modules" for Controllers, and each type of I/O system. Users optionally disable or extend these standard modules.

15 Users may elect several strategies for placing Device Alarm modules in Plant Areas including a small-system strategy, a "segregation" strategy and a "partitioned responsibility" strategy, for example. In the small system an "all in one" strategy is imposed in which the Plant Area concept is not applied. One or more Device Alarm Modules in each Controller form an integrated presentation of Device and SP88 Module alarms. In the segregation strategy a separate Plant Area is designated which has all Device Alarm modules
20 from all Controllers. The segregation strategy enables Area filtering/sorting on Alarm summaries, allowing Operators to focus on SP88 Module Alarms and Process/Maintenance Engineers to focus on Device Alarms. The partitioned responsibility strategy is used when the system becomes large enough to control scope of responsibility by Plant Area. Device Alarm Modules are placed in the same Plant Area as the SP88 modules impacted by the Device Alarm Modules. The partitioned responsibility strategy forms an integrated
25 presentation of Device and SP88 Module alarms for Plant Areas within scope of responsibility.

Alarms are supported for Device/Subsystem Attributes in Workstations. User-initiated applications such as Draw, View, engineering tools, and the like individually present information about abnormal conditions or errors encountered, in the appropriate context. Workstation Services which are activated on a workstation before a user logs on and continue to run through log-off/log-on cycles) implement a technique
30 for drawing attention to problems, even for an unattended Workstation. In one embodiment, Workstation Services are monitored by Device Alarm Modules executing in one or more Controllers. The Services construct a set of status and integrity Attributes, which are accessed by Controller(s) running instances of Device Alarm Modules which control the user-defined Device Alarming strategy. Pre-defined Control Module Definitions are supplied providing fairly comprehensive Device Alarm "modules" for Workstations,

and each major Service (e.g. communications, history journal, etc.) Users may disable or extend these standard modules.

In one embodiment of the process control system, event conditions that are not afforded "Alarm" status are given a priority of "LOG". LOG Alarms are not consolidated in ALARMS Attributes, do not appear in the Alarm Summary, do not provide audible annunciation of state changes, and appear in Event Journals as type "EVENT" rather than type "ALARM". In other respects a LOG priority Alarm operates as HIGH/MEDIUM/LOW priority Alarm so that an event (1) is enabled/disabled individually, (2) is disabled at the Module level with other alarms, (3) individual Alarms can be "turned into Events" by changing their priority to "LOG" (writing 3 to .PRI).

By setting .PRIAD at the Module level, one or more levels of Alarms can be converted to Events (e.g. writing 2 to .PRIAD drops all Alarm priorities in the Module by 2 steps; HIGH priority becomes LOW, MEDIUM and LOW priority become LOG, LOG remains LOG). Setting .PRIAD to 3 forces all Alarms in the Module to become "Events". CUALM, LAALM, NALM field supported to display current status (even blinking if unacked).

LOG events also (4) may invert the input boolean, to reverse the sense of the OK event condition for usage to log changes in state of Discrete Inputs, (5) may add user defined "alarm words" to the Alarm Types table to give event conditions useful names, (6) record all state changes for a LOG priority alarm in the Event Journal.

User selectable "intensities" of system event detection (e.g. "debug intensity", "normal intensity", "shutdown intensity") are configured in Control Module algorithms based on the setting of a user defined "intensity" Attribute. System Events are thus aligned with one (or perhaps more) Plant Areas, and so enabling the Event Journal on a Workstation for one or more Plant Areas automatically sets the destination for all "System Event" records.

A user changing an Attribute or invoking a method on a System Object is considered an event and is recorded in the appropriate Event Journal. Changes to control hierarchy (Control Module, Equipment Module, Plant Area, etc.) Attributes are recorded in the Event Journal(s) designated for that Plant Area. Changes to Device/Subsystem Attributes are recorded in the Event Journal(s) for the "primary Plant Area" designated for that Device.

Referring to FIGURE 42, a context diagram shows a context for defining an alarm event with respect to a control module. An Alarm appears in a "Plant Area scope" active alarm list presentation. A composite module (CM) instance 4210 includes a PID function block 4212, an output attribute 4214 and a high alarm attribute 4216. A user, such as a configuration engineer, selects the composite module 4210 for editing and selects "add alarm" 4220 on Attribute "HIHI". The user also selects the event priority definition

named "Advisory" 4232 and the event type definition named "HIALARM" 4234. The user then saves the changes to the control module 4210.

Referring again to FIGURE 40, alarm and event management is described. Alarms and user-defined "events" which are configured as LOG priority Alarms introduce multiple behaviors into the process control system.

A special kind of RtAttribute, hereafter called RtAlarmAttribute 4034, has a distinct "data type" supporting Alarm specific fields such as CUALM, NALM, and the like which may be read and written. Read access to the fields allows presentation of the state of individual Alarms. Write access to selected fields gives an ability to Acknowledge Alarms, and change certain alarm behaviors.

Several characteristics of Alarm behavior, such as Disabled and AUTOACK behavior, may be overridden on-line at the SP88 Module level. Reverting to the individual Alarm conditions of Enabled/Disabled and NoAutoAck/AutoAck occurs when the Module level overrides are removed. The changing of Module level overrides should "immediately" impact individual Alarm states. Thus changes on the Module level override force re-evaluation of all Alarm states within the module under the new conditions.

Active alarms are consolidated at the Module (RtModule 4030), Plant Area (RtPlantArea 4010), and User Session so that the "highest priority alarms" at each of the levels may be presented. This consolidation is accessed through the ALARMS[] Attribute supported by Module, Plant Area and User Session objects.

FLX™ Alarm Summary Links are used in FLX™ pictures (displays) to present a list of "current" alarms. The content for the Alarm Summary Link is maintained by the ALMSUM.EXE process (which shuts down when FLX™ shuts down, and FLX™ shuts down whenever an NT user logs off³, and NT users log off to allow a new user to "log in".) Thus the system must both "prime" the ALMSUM process with all current alarms (subject to current user responsibility scope) to get it started when a new user logs-on, and it must feed the ALMSUM process information about new alarm occurrences, and alarm acknowledgments so a up-to-date summary can be presented.

All alarm state transitions are directed to the appropriate Event Journal(s) (RtEventJournal 4020) for the Plant Area which "contains" the RtAlarmAttribute 4034. Multiple Event Journal targets are supported, so that a complete Event Journal can be reconstructed if one workstation running one of the Event Journals is off line for a period of time.

Audible annunciation of Alarm entry state transitions executes on workstations doing Alarm Summarization (feeding the FLX ALMSUM.EXE process). Audible annunciation may consist of a continuous tone (user configured frequency) on the PC speaker, and/or a continuous (looping) .WAV file played on a compatible sound card installed in the PC. A program is executed to turn off both the speaker

and the sound card, thus the sound may be turned off by any FIX™ (view button or keyboard) script, or an ICON in the program Group if FIX VIEW is not running.

Referring to FIGURE 43, an object communication diagram illustrates a method for performing an attribute write operation that evokes an "in alarm" status.

5 Previous to the attribute write operation, a RtAlarmAttribute has been configured and installed, alarms are ENABLED at both the Attribute and the Module level, alarm AUTOACK is false at both the Attribute and the Module level, and the current Alarm state is "Inactive/Acknowledged".

10 The "write Attribute" method causes the state of an Alarm Attribute to go into the alarm state. The alarm fields of the Alarm Attribute are updated to reflect the new Alarm state. An event occurrence record is constructed, and sent to the Module, Plant Area, and workstation applications (Alarm Summary, Event Journal), as needed.

15 When the attribute write operation is complete, the current Alarm state is "Active/Unacknowledged", the active alarm has been recorded by the Module, the event occurrence record has been constructed, and has been queued for transmission to devices monitoring this Plant Area in this Device.

20 In a step 4310 of the write attribute method, RtAlarmAttribute receives a writeAttribute, which causes a state transition in the boolean Attribute to enter the alarm state. In step 4312, RtAlarmAttribute gets alarmDisable status from the RtModule containing RtAlarmAttribute so that both Attribute and Module Alarm behavior are ENABLED. In step 4314, RtAlarmAttribute gets alarmAutoack status from the RtModule containing RtAlarmAttribute so that for both Attribute and Module Alarm AUTOACK is DISABLED. In step 4316, RtAlarmAttribute computes a new alarm state, the "Active/Unacknowledged" state and reads the prototype event descriptor string from RtEventTypeDefinition using AlarmType as an index. In step 4318, RtAlarmAttribute constructs the descriptionString for the RtEventOccurrenceRecord, reading current attribute values from the containing RtModule, if necessary. In step 4320, RtAlarmAttribute constructs a new RtEventOccurrenceRecord. Since a new alarm is created, RtAlarmAttribute tells its containing RtModule to addEventOccurrence in step 4322. In step 4324, RtModule tells its RtActiveAlarmList to addEventOccurrence, adding a new entry to the list and returning a handle by which the entry can be accessed in the future. This handle is ultimately stored by RtAlarmAttribute. In step 4326, RtModule sends the RtEventOccurrenceRecord to the RtPlantArea of RtModule via recordEventOccurrence. In step 4328, RtPlantArea tells its RtAreaEventLog to addEvent, RtAreaEventLog sees that at least one workstation client has registered an interest in receiving EventLog records, creates an RtEventLogRecord from the RtEventOccurrenceRecord, and queues the RtEventLogRecord, thereby destroying the RtEventOccurrenceRecord.

Also referring to **FIGURE 43**, the object communication diagram is also applicable to Acknowledgement of an Alarm, causing the current alarm state to go to "Active/Acknowledged". The new alarm state for the existing alarm is recorded by the Module, a new event occurrence record is constructed, and is been queued for transmission to devices monitoring this Plant Area in this Device. The method includes application of the "write Attribute" method (writing FALSE to the NALM Field) to cause the state of an Alarm Attribute to be acknowledged. Alarm fields of the Alarm Attribute are updated to reflect the new Alarm state. An event occurrence record is constructed, and sent to the Module, Plant Area, and workstation applications (Alarm Summary, Event Journal) as needed.

In step 4310, **RtAlarmAttribute** receives a **writeAttribute**, causing the NALM Field to change state to FALSE. In step 4312, **RtAlarmAttribute** gets **alarmDisable** status from the **RtModule** containing **RtAlarmAttribute** so that both Attribute and Module Alarm behavior are ENABLED. In step 4316, **RtAlarmAttribute** computes a new alarm state, the "Active/Unacknowledged" state and reads the prototype event descriptor string from **RtEventTypeDefinition** using **AlarmType** as an index. In step 4318, **RtAlarmAttribute** constructs the **descriptionString** for the **RtEventOccurrenceRecord**, reading current attribute values from the containing **RtModule**, if necessary. In step 4320, **RtAlarmAttribute** constructs a new **RtEventOccurrenceRecord**. Since an existing alarm is updated, **RtAlarmAttribute** tells the **RtModule** containing **RtAlarmAttribute** to **updateEventOccurrence**, identifying **RtModule** by the handle returned when from **updateEventOccurrence** in step 4322. In step 4324, **RtModule** tells **RtActiveAlarmList** to **updateEventOccurrence**, thereby updating the existing entry in the list. In step 4326, **RtModule** sends the **RtEventOccurrenceRecord** to the **RtPlantArea** of **RtModule** via **recordEventOccurrence**. In step 4328, **RtPlantArea** tells its **RtAreaEventLog** to **addEvent**, **RtAreaEventLog** sees that at least one workstation client has registered an interest in receiving **EventLog** records, creates an **RtEventLogRecord** from the **RtEventOccurrenceRecord**, and queues the **RtEventLogRecord**, thereby destroying the **RtEventOccurrenceRecord**.

Also referring to **FIGURE 43**, the object communication diagram is also applicable to acknowledgement of clearing of an alarm condition, in which the "write Attribute" method causes the state of an Alarm Attribute to go out of the alarm state. The alarm fields of the Alarm Attribute are updated to reflect the new Alarm state and an event occurrence record is constructed and sent to the Module, Plant Area, and workstation applications (Alarm Summary, Event Journal) as needed. When the write Attribute method is complete, the current Alarm state is "Inactive/Acknowledged". The current alarm information for this alarm has been removed by the Module. A new event occurrence record has been constructed and queued for transmission to devices monitoring this Plant Area in this Device.

In step 4310, **RtAlarmAttribute** receives a **writeAttribute**, which causes a state transition in the boolean Attribute to go out of the alarm state. In step 4312, **RtAlarmAttribute** gets **alarmDisable** status from the **RtModule** containing **RtAlarmAttribute** so that both Attribute and Module Alarm behavior are ENABLED. In step 4316, **RtAlarmAttribute** computes a new alarm state, the "Active/Unacknowledged"

state and reads the prototype event descriptor string from **RtEventTypeDefinition** using **AlarmType** as an index. In step 4318, **RtAlarmAttribute** constructs the **descriptionString** for the **RtEventOccurrenceRecord**, reading current attribute values from the containing **RtModule**, if necessary. In step 4320, **RtAlarmAttribute** constructs a new **RtEventOccurrenceRecord**. Since an existing alarm is cleared, 5 **RtAlarmAttribute** tells the **RtModule** containing **RtAlarmAttribute** to **updateEventOccurrence**, identifying **RtModule** by the handle returned when from **updateEventOccurrence**. In step 4322, **RtModule** tells **RtActiveAlarmList** to **updateEventOccurrence**, thereby updating the existing entry in the list. In step 4326, **RtModule** sends the **RtEventOccurrenceRecord** to the **RtPlantArea** of **RtModule** via **recordEventOccurrence**. In step 4328, **RtPlantArea** tells its **RtAreaEventLog** to **addEvent**, 10 **RtAreaEventLog** sees that at least one workstation client has registered an interest in receiving **EventLog** records, creates an **RtEventLogRecord** from the **RtEventOccurrenceRecord**, and queues the **RtEventLogRecord**, thereby destroying the **RtEventOccurrenceRecord**.

Also referring to **FIGURE 43**, the object communication diagram is also applicable to disablement of an alarm by causing the **ENAB** Field of an **Alarm Attribute** to become **FALSE**. The alarm fields of the 15 **Alarm Attribute** are updated to reflect the new **Alarm** state and an event occurrence record is constructed, and sent to the **Module**, **Plant Area**, and workstation applications (**Alarm Summary**, **Event Journal**), as needed. Following the disablement of an alarm, the current **Alarm** state is "Inactive/Acknowledged" and the **ENAB** field is **FALSE**. The current alarm information for this alarm has been removed by the **Module**. A new event occurrence record (alarm **DISABLE**) has been constructed and is queued for transmission to 20 devices monitoring this **Plant Area** in this **Device**.

In step 4310, **RtAlarmAttribute** receives a **writeAttribute**, which causes the **ENAB** Field to become **FALSE**. In step 4316, **RtAlarmAttribute** computes a new alarm state, the "Active/Unacknowledged" state and reads the prototype event descriptor string from **RtEventTypeDefinition** using **AlarmType** as an index. In step 4318, **RtAlarmAttribute** constructs the **descriptionString** for the 25 **RtEventOccurrenceRecord**, reading current attribute values from the containing **RtModule**, if necessary. In step 4320, **RtAlarmAttribute** constructs a new **RtEventOccurrenceRecord**, identifying this event as an alarm disable event. Since the alarm is disabled when previously active, this event is the clearing of an existing alarm, **RtAlarmAttribute** tells the **RtModule** containing **RtAlarmAttribute** to **clearEventOccurrence**, identifying **RtModule** by the handle returned when from **clearEventOccurrence**. 30 In step 4322, **RtModule** tells **RtActiveAlarmList** to **clearEventOccurrence**, thereby removing the existing entry from the list. In step 4326, **RtModule** sends the **RtEventOccurrenceRecord** to the **RtPlantArea** of **RtModule** via **recordEventOccurrence**. In step 4328, **RtPlantArea** tells its **RtAreaEventLog** to **addEvent**, **RtAreaEventLog** sees that at least one workstation client has registered an interest in receiving **EventLog** records, creates an **RtEventLogRecord** from the **RtEventOccurrenceRecord**, and queues the 35 **RtEventLogRecord**, thereby destroying the **RtEventOccurrenceRecord**.

Also referring to FIGURE 43, the object communication diagram is also applicable to enablement of an alarm by causing the ENAB Field of an Alarm Attribute to become TRUE. Prior to enablement of an alarm, the current state of the boolean Attribute shows the alarm would be Active if Enabled, AutoAck is False at the Attribute and Module level. The alarm fields of the Alarm Attribute are updated to reflect the new Alarm state and an event occurrence record is constructed, and sent to the Module, Plant Area, and workstation applications (Alarm Summary, Event Journal), as needed. Following the disablement of an alarm, the current Alarm state is "Active/Unacknowledged" and the ENAB field is TRUE. The current alarm information for this alarm is stored by the Module. A new event occurrence record (alarm Active/Unacknowledged) has been constructed and is queued for transmission to devices monitoring this Plant Area in this Device.

In step 4310, RtAlarmAttribute receives a writeAttribute, which causes the ENAB Field to become TRUE. In step 4312, RtAlarmAttribute gets alarmDisable status from the RtModule containing RtAlarmAttribute so that both Attribute and Module Alarm behavior are ENABLED. In step 4314, RtAlarmAttribute gets alarmAutoack status from the RtModule containing RtAlarmAttribute so that for both Attribute and Module Alarm AUTOACK is DISABLED. In step 4316, RtAlarmAttribute computes a new alarm state, the "Active/Unacknowledged" state and reads the prototype event descriptor string from RtEventTypeDefinition using AlarmType as an index. In step 4318, RtAlarmAttribute constructs the descriptionString for the RtEventOccurrenceRecord, reading current attribute values from the containing RtModule, if necessary. In step 4320, RtAlarmAttribute constructs a new RtEventOccurrenceRecord. Since the alarm is new, RtAlarmAttribute tells the RtModule containing RtAlarmAttribute to addEventOccurrence, identifying RtModule by the handle returned when from addEventOccurrence. In step 4322, RtModule tells RtActiveAlarmList to addEventOccurrence, thereby adding a new entry to the list. In step 4326, RtModule sends the RtEventOccurrenceRecord to the RtPlantArea of RtModule via recordEventOccurrence. In step 4328, RtPlantArea tells its RtAreaEventLog to addEvent, RtAreaEventLog sees that at least one workstation client has registered an interest in receiving EventLog records, creates an RtEventLogRecord from the RtEventOccurrenceRecord, and queues the RtEventLogRecord, thereby destroying the RtEventOccurrenceRecord.

While the invention has been described with reference to various embodiments, it will be understood that these embodiments are illustrative and that the scope of the invention is not limited to them. Many variations, modifications, additions and improvements of the embodiments described are possible.

WE CLAIM:

1 1. A process control system (100) for controlling a plurality of field devices of multiple different
2 field device types including smart-type field devices (132) and non-smart-type field devices (136), the process
3 control system comprising:

4 a plurality of distributed controllers (110) coupled to the field devices; and

5 a software system including a plurality of control modules (440) selectively installable and operative
6 on ones of the plurality of distributed controllers, the software system for communicating
7 with the smart-type and the non-smart-type field devices and for controlling the smart-type
8 and the non-smart-type field devices.

1 2. A process control system according to Claim 1 wherein the software system further includes:

2 a configuration program for configuring the control modules and installing the control modules on
3 the plurality of distributed controllers, the distributed controllers retaining the configuration
4 until reconfigured.

1 3. A process control system according to Claim 1 wherein the plurality of control modules is
2 configured into a communication services hierarchy including:

3 a remote object communications (ROC) level for communicating messages between two control
4 modules in a same controller and between two control modules in different controllers, and
5 a low level communications level for interfacing with communications hardware and transmitting
6 messages across the communications hardware.

1 4. A process control system (100) for controlling a plurality of field devices of multiple different
2 field device types including smart-type field devices (132) and non-smart-type field devices (136), the process
3 control system comprising:

4 a plurality of distributed controllers (110) coupled to the field devices; and

5 a plurality of control means selectively installable and operative on ones of the plurality of
6 distributed controllers for, in combination, controlling the smart-type and the non-smart-
7 type field devices.

1 5. A process control system (100) comprising:

2 a plurality of field devices of multiple different field device types including smart-type field devices
3 (132) and non-smart-type field devices (136);

4 a plurality of distributed controllers (110) coupled to the field devices;

5 a workstation (102) coupled to the distributed controllers; and

6 a plurality of control means selectively installable and operative on ones of the plurality of
7 distributed controllers for, in combination, controlling the smart-type and the non-smart-
8 type field devices.

1 6. A computer program product comprising:

2 a computer usable medium having computable readable code embodied therein including a process
3 control software system for controlling a plurality of field devices of multiple different field
4 device types including smart-type field devices (132) and non-smart-type field devices
5 (136), the process control system (100) including a plurality of distributed controllers (110)
6 coupled to the field devices, the executable program code including:
7 a software system including a plurality of control modules (440) selectively installable and
8 operative on ones of the plurality of distributed controllers; and
9 a communication and control routine for communicating with the smart-type and the non-
10 smart-type field devices and for controlling the smart-type and the non-smart-type
11 field devices.

1 7. A computer program product according to Claim 6, the executable program code further
2 comprising:

3 a user interface (300) for interfacing with a user; wherein:
4 the smart-type and the non-smart-type field devices operate in compliance with defined bus-based
5 architecture standards and the software system performs smart-type control operations and
6 non-smart-type control operations transparent to the user over the user interface.

1 8. A computer program product according to Claim 6 wherein the software system performs smart-
2 type control operations using the plurality of control modules distributed on the plurality of distributed
3 controllers operating on the smart-type field devices and non-smart-type operations operating on the non-
4 smart-type field devices independently, simultaneously and in parallel.

1 9. A computer program product according to Claim 6 wherein the software system further includes:
2 a configuration program for configuring the control modules and installing the control modules on
3 the plurality of distributed controllers, the distributed controllers retaining the configuration
4 until reconfigured.

1 10. A computer program product according to Claim 6 wherein the plurality of control modules is
2 configured into a communication services hierarchy including:

3 a remote object communications (ROC) level for communicating messages between two control
4 modules in a same controller and between two control modules in different controllers, and
5 a low level communications level for interfacing with communications hardware and transmitting
6 messages across the communications hardware.

1 11. A process control system (100) for controlling a plurality of field devices of multiple different
2 field device types including standard-protocol field devices (6) and non-standard-protocol field devices (12),
3 the process control system comprising:

4 a plurality of distributed controllers (110) coupled to the field devices; and
5 a software system including a plurality of function blocks (522) defined in a standard protocol, the
6 function blocks being selectively installable and operative on ones of the plurality of
7 distributed controllers for selectively controlling a standard-protocol field device and a non-
8 standard-protocol field device.

1 12. A process control system (100) for controlling a plurality of field devices of multiple different
2 field device types including standard-protocol field devices (6) and non-standard-protocol field devices (12),
3 the process control system comprising:

4 a plurality of distributed controllers (110) coupled to the field devices; and
5 a plurality of control means selectively installable and operative on ones of the plurality of
6 distributed controllers controlling, in combination, the standard-protocol field devices and
7 the non-standard-protocol field devices.

1 13. A process control system (100) comprising:

2 a plurality of field devices of multiple different field device types including standard-protocol field
3 devices (6) and non-standard-protocol field devices (12);
4 a plurality of distributed controllers (110) coupled to the field devices;
5 a workstation (102) coupled to the distributed controllers; and
6 a software system including a plurality of function blocks (522) defined in a standard protocol, the
7 function blocks being selectively installable and operative on ones of the plurality of
8 distributed controllers for selectively controlling a standard-protocol field device and a non-
9 standard-protocol field device.

1 14. A process control system (100) for controlling a plurality of field devices of multiple different
2 field device types including standard-protocol field devices (6) and non-standard-protocol field devices (12),
3 the process control system comprising:

4 a plurality of distributed controllers (110) coupled to the field devices; and
5 a software system including a plurality of function blocks (522) defined in a standard protocol, the
6 function blocks being selectively definable, creatable, modifiable, and installable as a
7 control module that is operative on ones of the plurality of distributed controllers for
8 selectively controlling a standard-protocol field device and a non-standard-protocol field
9 device.

1 15. A process control system (100) comprising:

2 a plurality of field devices of multiple different field device types including standard-protocol field
3 devices (6) and non-standard-protocol field devices (12);

4 a plurality of distributed controllers (110) coupled to the field devices;

5 a workstation (102) coupled to the distributed controllers; and

6 a software system including a plurality of function blocks (522) defined in a standard protocol, the
7 function blocks being selectively definable, creatable, modifiable, and installable as a
8 control module that is operative on ones of the plurality of distributed controllers for
9 selectively controlling a standard-protocol field device and a non-standard-protocol field
10 device.

1 16. A process control system (100) for controlling a plurality of field devices of multiple different
2 field device types including standard-protocol field devices (6) and non-standard-protocol field devices (12),
3 the process control system comprising:

4 a plurality of distributed controllers (110) coupled to the field devices; and

5 a plurality of control modules (440) selectively definable, creatable, modifiable, and installable and
6 operative on ones of the plurality of distributed controllers controlling, in combination, the
7 standard-protocol field devices and the non-standard-protocol field devices.

1 17. A process control system according to any of Claims 11, 12, 13, 14, 15 and 16 further
2 comprising:

3 a user interface (300) for interfacing with a user; wherein:

4 the standard-protocol field devices and the non-standard-protocol field devices operate in
5 compliance with the defined Fieldbus architecture standard and the software system
6 performs Standard-protocol control operations on standard-protocol field devices and non-
7 standard-protocol field devices transparent to the user over the user interface.

1 18. A process control system according to any of Claims 11, 12, 13, 14, 15 and 16 wherein the
2 software system further includes:

3 a configuration program for configuring the function blocks and installing the function blocks on the
4 plurality of distributed controllers, the distributed controllers retaining the configuration
5 until reconfigured.

1 19. A process control system according to any of Claims 11, 12, 13, 14, 15 and 16 wherein the
2 plurality of function blocks is configured into a communication services hierarchy including:

3 a remote object communications (ROC) level for communicating messages between two function
4 blocks in a same controller and between two function blocks in different controllers, and

5 a low level communications level for interfacing with communications hardware and transmitting
6 messages across the communications hardware.

1 20. A computer program product comprising:

2 a computer usable medium having computable readable code embodied therein including a process
3 control system (100) for controlling a plurality of field devices of multiple different field
4 device types including standard-protocol field devices (6) and non-standard-protocol field
5 devices (12), the process control system including a plurality of distributed controllers (110)
6 coupled to the field devices, the executable program code comprising:

7 a software system including a plurality of function blocks (522) defined in a standard protocol, the
8 function blocks being selectively installable and operative on ones of the plurality of
9 distributed controllers for selectively controlling a standard-protocol field device and a non-
10 standard-protocol field device.

1 21. A process control system (100) comprising:

2 a field device;

3 a controller (110) coupled to the field device;

4 a workstation (102) coupled to the controller, the workstation including a user interface (300); and

5 a software system implementing a control strategy for the process control system, the control
6 strategy being selectively defined, created, modified, and apportioned via the user interface
7 into a plurality of control strategy modules, and the plurality of control strategy modules
8 being selectively distributed among the field device, controller and workstation, the control
9 strategy modules operating mutually independently and in parallel.

1 22. A process control system (100) comprising:

2 a field device;

3 a control means coupled to the field device for controlling the field device;

4 an interface means coupled to the control means for interfacing the control process system to a user;
5 and

6 a control strategy means for defining, creating, modifying, and implementing a process control
7 strategy under direction of the user, the user selectively apportioning the control strategy
8 means into a plurality of control strategy modules, and the user selectively distributing the
9 control strategy modules among the field device, control means and interface means, the
10 control strategy modules operating mutually independently and in parallel.

1 23. A process control system according to either Claim 21 or Claim 22, wherein:

2 the software system further includes a user interface for interfacing the process control system to a
3 user, and

4 the control strategy is selectively defined, created, modified, and apportioned, and the plurality of
5 control strategy modules selectively distributed by the user.

1 24. A process control system according to either Claim 21 or Claim 22, wherein:
2 the field device includes a control element; and
3 the software system includes a control strategy module that is distributed to the field device control
4 element.

1 25. A process control system according to either Claim 21 or Claim 22 wherein:
2 the field device is a Fieldbus standard device including a control element; and
3 a control strategy module is distributed to the control element and operates selectively as a Fieldbus
4 standard function block or a custom, user-defined control module.

1 26. A process control system according to either Claim 21 or Claim 22 wherein the software system
2 further includes:
3 a configuration program for defining, creating and configuring the control strategy modules and
4 installing the control strategy modules among the field device, controller and workstation,
5 the distributed controllers retaining the configuration until reconfigured.

1 27. A process control system according to either Claim 21 or Claim 22, wherein:
2 the control strategy modules are selectively defined, modified, and created by a user creating custom
3 control strategy modules; and
4 the control strategy modules are selectively distributed by transferring a selected control strategy
5 module to a selected one of the field device, controller and workstation.

1 28. A process control system according to either Claim 21 or Claim 22 wherein the plurality of
2 control strategy modules are configured into a communication services hierarchy including:
3 a remote object communications (ROC) level for communicating messages between two control
4 strategy modules in a same controller and between two control strategy modules in different
5 controllers, and
6 a low level communications level for interfacing with communications hardware and transmitting
7 messages across the communications hardware.

1 29. A method of operating a process control system (100) including a distributed controller (110)
2 and a distributed field device comprising the steps of:
3 executing process control operations;
4 executing an editor program during execution of the process control system operations;
5 using the editor program, defining a control strategy including the steps of :

6 building a plurality of function blocks (522) and control modules (440); and
7 downloading user-specified function blocks and control modules selectively among the
8 distributed controller and the distributed field device;
9 executing the function blocks and control modules distributed to the controller and distributed to the
10 field device mutually independently and in parallel.

1 30. A computer program product for use in a process control system (100) including a field device, a
2 controller (110) coupled to the field device, and a workstation (102) coupled to the controller, the computer
3 program product comprising:

4 a computer usable medium having computable readable code embodied therein including a software
5 system implementing a control strategy for the process control system, the control strategy
6 being selectively definable, creatable, modifiable, and apportionable into a plurality of
7 control strategy modules, and the plurality of control strategy modules being selectively
8 distributable among the field device, controller and workstation, the control strategy
9 modules operating mutually independently and in parallel.

1 31. A process control system (100) comprising:

2 a field device including a source of diagnostic information;

3 a controller (110) coupled to the field device;

4 a workstation (102) coupled to the controller and including a user interface (300); and

5 a software system implementing a diagnostic monitoring and display program for the process control
6 system, the diagnostic monitoring and display program including:

7 a plurality of diagnostic modules selectively defined and created via the user interface for
8 access using the diagnostic monitoring and display program, the plurality of
9 diagnostic modules upon creation being selectively distributed among the field
10 device, the controller and the workstation, the diagnostic modules operating
11 mutually independently and in parallel accessing the source of diagnostic
12 information; and

13 a display routine for accessing diagnostic information from the plurality of diagnostic
14 modules and displaying the diagnostic information accessed from the plurality of
15 diagnostic modules uniformly for all diagnostic modules in the process control
16 system so that the diagnostic information relating to a process that operates both in
17 the controller and in the field device is displayed in the same manner regardless of
18 the source of the diagnostic information.

1 32. A process control system (100) comprising:

2 a field device including a source of diagnostic information;

3 a controller (110) coupled to the field device;

4 a workstation (102) coupled to the controller and including a user interface (300); and
5 a software system implementing a diagnostic monitoring and display program for the process control
6 system, the diagnostic monitoring and display program including:
7 a plurality of control modules (440) for selectively implementing a process control strategy,
8 a plurality of diagnostic modules selectively defined and created via the user interface for
9 access using the diagnostic monitoring and display program, the plurality of
10 diagnostic modules upon creation being selectively distributed among the field
11 device, the controller and the workstation, the diagnostic modules operating
12 mutually independently and in parallel accessing the source of diagnostic
13 information; and
14 a display routine for accessing diagnostic information from the plurality of diagnostic
15 modules and a control scheme from the plurality of control modules and for
16 respectively displaying the diagnostic information accessed from the plurality of
17 diagnostic modules and the control strategy accessed from the plurality of control
18 modules so that the diagnostic information and the control information are
19 accessed in the same manner.

1 33. A process control system (100) comprising:
2 a plurality of field devices, a field device of the plurality of field devices including a source of
3 diagnostic information;
4 a plurality of controllers (110), a controller of the plurality of controllers being coupled to a field
5 device of the plurality of field devices;
6 a workstation (102) coupled to the controller and including a user interface (300); and
7 a software system implementing a diagnostic monitoring and display program for the process control
8 system, the diagnostic monitoring and display program including:
9 a plurality of diagnostic modules selectively defined and created via the user interface for
10 access using the diagnostic monitoring and display program, the plurality of
11 diagnostic modules upon creation being selectively distributed among the field
12 devices, the controllers and workstation, the diagnostic modules operating
13 mutually independently and in parallel; and
14 a display routine for accessing diagnostic information from the plurality of diagnostic modules
15 operating mutually independently on the field devices, the controllers and the workstation
16 and displaying the diagnostic information accessed from the plurality of diagnostic modules
17 uniformly so that the diagnostic information relating to a process that operates more than
18 one of the field devices, the controllers and the workstation is displayed as being generated
19 at a single location.

1 34. A process control system (100) comprising:
2 a plurality of field devices, a field device of the plurality of field devices including a source of
3 diagnostic information;
4 a plurality of control means for controlling a field device, a control means of the plurality of control
5 means being coupled to a field device of the plurality of field devices;
6 an interface means coupled to the plurality of control means for interfacing the control process
7 system to a user;
8 a diagnostic means for implementing a process control strategy, the diagnostic means being
9 selectively defined and created as a plurality of diagnostic modules, the plurality of
10 diagnostic modules upon creation being selectively distributed among the field device,
11 control means and interface means, the diagnostic modules operating mutually
12 independently and in parallel; and
13 a display means for accessing diagnostic information from the plurality of diagnostic means
14 operating mutually independently on the field devices, the control means and the interface
15 means and displaying the diagnostic information accessed from the plurality of diagnostic
16 modules uniformly so that the diagnostic information relating to a process that operates
17 more than one of the field devices, the control means and the interface means is displayed
18 as being generated at a single location.

1 35. A process control system (100) comprising:
2 a field device including a source of diagnostic information;
3 a controller (110) coupled to the field device;
4 a workstation (102) coupled to the controller and including a user interface (300); and
5 a software system implementing a diagnostic monitoring and display program for the process control
6 system, the diagnostic monitoring and display program including:
7 a control strategy for the process control system, the control strategy being selectively
8 apportioned into a plurality of control strategy modules and selectively distributed
9 among the field device, controller and workstation, the control strategy modules
10 operating mutually independently and in parallel;
11 a plurality of diagnostic modules selectively defined and created via the user interface for
12 access using the diagnostic monitoring and display program, the plurality of
13 diagnostic modules upon creation being selectively distributed among the field
14 device, the controller and the workstation, the diagnostic modules operating
15 mutually independently and in parallel accessing the source of diagnostic
16 information; and

17 a display routine for accessing diagnostic information from the plurality of diagnostic
18 modules and displaying the control strategy and the diagnostic information
19 accessed from the plurality of diagnostic modules uniformly so that the diagnostic
20 information relating to a process that operates both in the controller and in the
21 field device is displayed as being generated at a single location.

1 36. A process control system according to any of Claims 31, 32, 33, 34, and 35 wherein:
2 the plurality of diagnostic modules are selectively defined and created, and selectively distributed by
3 a user.

1 37. A process control system according to any of Claims 31, 32, 33, 34, and 35, wherein:
2 the field device includes a control element; and
3 a diagnostic module of the plurality of diagnostic modules is distributed to the field device and
4 monitors a condition or status of the control element.

1 38. A process control system according to any of Claims 31, 32, 33, 34, and 35, further comprising:
2 a network coupled to the controller; and
3 an external node coupled to the controller through the network so that device information including
4 real-time data, history information, event statistics, configuration data and diagnostic
5 information are accessed using network standard communications.

1 39. A computer program product comprising:
2 a computer usable medium having computable readable code embodied therein for controlling a
3 process control system (100) including a source of diagnostic information, a controller
4 coupled to the field device, and a workstation (102) coupled to the controller and including
5 a user interface (300), the executing program code implementing a diagnostic monitoring
6 and display program for the process control system, the diagnostic monitoring and display
7 program including:
8 a plurality of diagnostic modules selectively defined and created via the user interface for
9 access using the diagnostic monitoring and display program, the plurality of
10 diagnostic modules upon creation being selectively distributed among the field
11 device, the controller and the workstation, the diagnostic modules operating
12 mutually independently and in parallel accessing the source of diagnostic
13 information; and
14 a display routine for accessing diagnostic information from the plurality of diagnostic
15 modules and displaying the diagnostic information accessed from the plurality of
16 diagnostic modules uniformly for all diagnostic modules in the process control
17 system so that the diagnostic information relating to a process that operates both in

the controller and in the field device is displayed in the same manner regardless of the source of the diagnostic information.

40. A computer program product comprising:

a computer usable medium having computable readable code embodied therein for controlling a process control system (100) including a field device having a source of diagnostic information, a controller coupled to the field device, a workstation (102) coupled to the controller, and a user interface (300), the computable readable code implementing a diagnostic monitoring and display program for the process control system, the diagnostic monitoring and display program including:

a plurality of control modules (440) for selectively implementing a process control strategy;

a plurality of diagnostic modules selectively defined and created via the user interface for access using the diagnostic monitoring and display program, the plurality of diagnostic modules upon creation being selectively distributed among the field device, the controller and the workstation, the diagnostic modules operating mutually independently and in parallel accessing the source of diagnostic information; and

a display routine for accessing diagnostic information from the plurality of diagnostic modules and a control scheme from the plurality of control modules and for respectively displaying the diagnostic information accessed from the plurality of diagnostic modules and the control strategy accessed from the plurality of control modules so that the diagnostic information and the control information are accessed in the same manner.

41. A control system for controlling a process comprising:

a controller coupled to the process; and

a software system executing on the controller and implementing a control strategy for controlling the process, the control strategy being defined by a layered hierarchy of modules including elemental modules containing exclusively one or more primitives and composite modules.

42. A process control system (100) for controlling a plurality of field devices, the process control system comprising:

a plurality of distributed controllers (110) coupled to the field devices for controlling a process; and

a distributed software system executing on the plurality of distributed controllers and implementing a control strategy for controlling the process, the control strategy being defined by a layered hierarchy of modules distributed for execution among the plurality of distributed controllers, the hierarchy of modules including elemental modules containing exclusively one or more primitives and composite modules.

1 43. A process control system (100) comprising:
2 a plurality of field devices;
3 a plurality of distributed controllers (110) coupled to the field devices for controlling a process; and
4 a distributed software system executing on the plurality of distributed controllers and implementing
5 a control strategy for controlling the process, the control strategy being defined by a layered
6 hierarchy of modules distributed for execution among the plurality of distributed controllers
7 and the plurality of field devices, the hierarchy of modules including elemental modules
8 containing exclusively one or more primitives and composite modules.

1 44. A control system for controlling a process under direction of a user, the control system
2 comprising:
3 a controller (110) coupled to the process; and
4 a software system executing on the controller and implementing a control strategy for controlling the
5 process, the control strategy being defined by a plurality of control modules (440) which are
6 objects of a container class, a control module of the plurality of control modules having a
7 specified task and a predefined external interface, the control module being encapsulated in
8 the software system and accessed through the predefined external interfaces so that the
9 control module is user-modifiable.

1 45. A control system according to any of Claims 41, 42, 43, and 44 wherein:
2 the process includes a field device; and
3 an elemental module is an elemental function block defined in accordance with a Fieldbus standard
4 protocol.

1 46. A control system according to any of Claims 41, 42, 43, and 44 further comprising:
2 a user interface coupled to the controller for specifying the control strategy, the control strategy
3 being user-specified by a module type of constituent elemental modules and composite
4 modules, by interconnections between the constituent modules and by input/output
5 connections of the constituent modules.

1 47. A control system according to Claim 46 wherein:
2 the user interface for modifying the control strategy is implemented in a plurality of process control
3 programming languages.

1 48. A control system according to any of Claims 41, 42, 43, and 44 wherein the software system
2 further includes:

3 a configuration program for defining the control strategy and installing the control strategy on the
4 controller, the controller retaining the configuration until reconfigured.

1 49. A computer program product comprising:

2 a computer usable medium having computable readable code embodied therein for controlling a
3 process control system (100) for controlling a plurality of field devices, the process control
4 system including a plurality of distributed controllers (110) coupled to the field devices for
5 controlling a process, the computable readable code including:

6 a distributed software system executing on the plurality of distributed controllers and implementing
7 a control strategy for controlling the process, the control strategy being defined by a layered
8 hierarchy of modules distributed for execution among the plurality of distributed
9 controllers, the hierarchy of modules including elemental modules containing exclusively
10 one or more primitives and composite modules.

1 50. A computer program product comprising:

2 a computer usable medium having computable readable code embodied therein for controlling a
3 process control system (100) including a plurality of field devices and a plurality of
4 distributed controllers (110) coupled to the field devices for controlling a process, the
5 computable readable code including:

6 a distributed software system executing on the plurality of distributed controllers and implementing
7 a control strategy for controlling the process, the control strategy being defined by a layered
8 hierarchy of modules distributed for execution among the plurality of distributed controllers
9 and the plurality of field devices, the hierarchy of modules including elemental modules
10 containing exclusively one or more primitives and composite modules.

1 51. A process control system (100) comprising:

2 a process;

3 a plurality of controllers (110) coupled to the process;

4 a workstation (102) coupled to the plurality of controllers and including a user interface (300); and

5 a software system including a network operating system and implementing a routine for
6 automatically sensing a connection of a controller to a network and incorporating the
7 controller into the network operating system.

1 52. A process control system according to Claim 51 wherein the software system further comprises:

2 a software routine responsive to automatic sensing of a connection of a controller to the network for
3 automatically configuring an input/output (I/O) subsystem in a control system.

1 53. A process control system according to Claim 51, wherein the routine for automatically sensing a
2 connection of a controller to a network and incorporating the controller into a network operating system
3 comprises:

4 means for connecting a controller to the network;

5 means operative in the connected controller for sending a request to confirm a network address
6 assignment, the request being accompanied by the controller media access control (MAC)
7 address;

8 a network configuration service including:

9 means for receiving the request to confirm;

10 means for searching a table of configured devices for a matching MAC address;

11 means operative when the MAC address matches for generating device and network
12 information including a network address from a device table;

13 means operative when the MAC address does not match for generating device and network
14 information including a network address from MAC address-based default

15 information and adding the default information to the device table; and

16 means operative when the MAC address does not match for assigning the connected controller
17 under user control either as a new device added to the device table or as a device
18 configuration previously existing in the device table.

1 ~~54. A method of automatically sensing a connection of a controller to a network and incorporating~~
2 ~~the controller into a network operating system including the steps of:~~

3 connecting a controller to the network;

4 sending, by the connected controller, a request to confirm a network address assignment, the
5 request being accompanied by the controller media access control (MAC) address;

6 receiving, by a network configuration service, the request to confirm and responding by performing
7 the steps of:

8 searching a table of configured devices for a matching MAC address;

9 when the MAC address matches, generating device and network information including a
10 network address from a device table;

11 when the MAC address does not match, generating device and network information
12 including a network address from MAC address-based default information and
13 adding the default information to the device table; and

14 when the MAC address does not match, assigning the connected controller under user control either
15 as a new device added to the device table or as a device configuration previously existing in
16 the device table.

1 55. A method of automatically configuring an input/output (I/O) subsystem in a control system
2 comprising the steps of:
3 interrogating an I/O Card at a user-specified card position to determine a Card Type and a number
4 of I/O Ports in the I/O Card;
5 determining whether the interrogated I/O Card is previously defined in an engineering database;
6 if the I/O Card is not previously defined in the engineering database, defining an I/O Card of a
7 suitable type and I/O Ports of a suitable number, the suitable type and number being
8 predetermined for the card position;
9 interrogating the I/O Ports of an I/O Card in accordance with the Card Type to determine a Port
10 Type and a number of I/O Devices on the I/O Port;
11 if the I/O Port is not previously defined in the engineering database for the port address, defining an
12 I/O Port of a suitable type and I/O Devices of a suitable number, the suitable type and
13 number being predefined;
14 interrogating the I/O Devices in accordance with the Port Type to determine a Device Type;
15 if the I/O Device is not previously defined in the engineering database for the device address,
16 defining an I/O Device of a suitable type, the suitable type being predefined; and
17 creating instrument signal tags (ISTs) for primary signal sources on the I/O Ports and the I/O
18 Devices.

1 56. A method according to Claim 55, further comprising the steps of:
2 determining whether an I/O Card exists in the engineering database for the card position;
3 if the I/O Card exists in the engineering database, determining whether the Card Type in the
4 engineering database matches the Card Type sensed at the card position,
5 if the Card Type in the engineering database does not match the Card Type sensed at the card
6 position executing the steps of:
7 generating a graphic notification of the mismatch;
8 interrogating a user to determine whether the engineering database is to be changed to
9 include the sensed Card Type; and
10 changing the Card Type in the engineering database to the sensed Card Type if requested by the
11 user.

1 57. A method according to Claim 55, further comprising the steps of:
2 determining whether an I/O Port exists in the engineering database for the port address;
3 if the I/O Port exists in the engineering database, determining whether the Port Type in the
4 engineering database matches the type of the sensed I/O Port sensed at the port address;
5 if the Port Type in the engineering database does not match the type of the sensed I/O Port executing
6 the steps of:

7 requesting advisement of the user to determine whether the engineering database is to be
8 updated to match the sensed I/O Port; and
9 changing the Port Type in the engineering database to the sensed Port Type if requested by
10 the user.

1 58. A method according to Claim 55, further comprising the steps of:
2 determining whether an I/O Device exists in the engineering database for the device address;
3 if the I/O Device exists in the engineering database, determining whether the Device Type in the
4 engineering database matches the type of the sensed I/O Device sensed at the device
5 address;
6 if the Device Type in the engineering database does not match the type of the sensed I/O Device
7 executing the steps of:
8 requesting advisement of the user to determine whether the engineering database is to be
9 updated to match the sensed I/O Device; and
10 changing the Device Type in the engineering database to the sensed Device Type if
11 requested by the user.

1 59. A computer program product comprising:
2 a computer usable medium having computable readable code embodied therein for automatically
3 sensing a connection of a controller to a network and incorporating the controller into a
4 network operating system including:
5 a routine for connecting a controller to the network;
6 a routine for sending, by the connected controller, a request to confirm a network address
7 assignment, the request being accompanied by the controller media access control
8 (MAC) address;
9 a routine for receiving, by a network configuration service, the request to confirm and
10 responding by executing routines including:
11 a routine for searching a table of configured devices for a matching MAC address;
12 a routine for determining when the MAC address matches, and for a matching
13 address generating device and network information including a network
14 address from a device table;
15 a routine for determining when the MAC address does not match, and for a
16 nonmatching address generating device and network information
17 including a network address from MAC address-based default
18 information and adding the default information to the device table; and
19 a routine for determining when the MAC address does not match, and for a
20 nonmatching address assigning the connected controller under user

21 control either as a new device added to the device table or as a device
22 configuration previously existing in the device table.

1 60. A computer program product comprising:

2 a computer usable medium having computable readable code embodied therein for automatically
3 configuring an input/output (I/O) subsystem in a control system comprising:
4 a routine for interrogating an I/O Card at a user-specified card position to determine a Card
5 Type and a number of I/O Ports in the I/O Card;
6 a routine for determining whether the interrogated I/O Card is previously defined in an
7 engineering database;
8 a routine operative when the I/O Card is not previously defined in the engineering database
9 and defining an I/O Card of a suitable type and I/O Ports of a suitable number, the
10 suitable type and number being predetermined for the card position;
11 a routine for interrogating the I/O Ports of an I/O Card in accordance with the Card Type to
12 determine a Port Type and a number of I/O Devices on the I/O Port;
13 a routine operative when the I/O Port is not previously defined in the engineering database
14 for the port address, the routine defining an I/O Port of a suitable type and I/O
15 Devices of a suitable number, the suitable type and number being predefined;
16 a routine for interrogating the I/O Devices in accordance with the Port Type to determine a
17 Device Type;
18 a routine operative when the I/O Device is not previously defined in the engineering
19 database for the device address, the routine defining an I/O Device of a suitable
20 type, the suitable type being predefined; and
21 a routine for creating instrument signal tags (ISTs) for primary signal sources on the I/O
22 Ports and the I/O Devices.

1 61. A process control system (100) comprising:

2 a process;
3 a plurality of devices coupled to the process;
4 a communication network coupled to the devices;
5 a workstation (102) coupled to the plurality of devices via the network and including a user interface
6 (300); and
7 a software system executable on the network and implementing a routine for automatically sensing a
8 connection of a device to a network and placing the connected device in an accessible state
9 for communicating with a user via the user interface.

1 62. A control system comprising:

2 a network;

3 a plurality of devices coupled to the network;
4 a distributed controller coupled to the plurality of devices and controlling the plurality of devices
5 according to a defined control configuration, the distributed controller including:
6 a control logic for sensing a device that is connected to the network but not included in the
7 defined control configuration;
8 a control logic for supplying initial interconnect information to the connected device; and
9 a control logic for uploading configuration parameters from the connected device to the
10 distributed controller.

1 63. A process control system according to either Claim 61 or Claim 62 wherein the software system
2 further comprises:

3 a routine for configuring the connected device in a network control configuration of the plurality of
4 devices.

1 64. A process control system according to Claim 63 wherein the routine for configuring the
2 connected device further comprises:

3 a user-interactive routine for determining a device type of the connected device;
4 a user-interactive routine for determining a role of the connected device with respect to the process
5 control system;
6 a user-interactive routine for assigning a physical device tag the determined role; and
7 a user-interactive routine for verifying connection of the device to the network.

1 65. A process control system according to Claim 63 wherein the routine for configuring the
2 connected device further comprises:

3 a user-interactive routine for initiating calibrating the connected device; and
4 a user-interactive routine for configuring the device within an overall control scheme of the process
5 control system.

1 66. A process control system according to either Claim 61 or Claim 62 wherein the software system
2 further comprises:

3 a routine for commissioning the connected device including:
4 a user-interactive routine for assigning a physical device tag, a device address, and a device
5 identification to the connected device; and
6 a user-interactive routine for installing a control strategy to the digital device.

67. A method of configuring a control system comprising:
predetermining a configuration of devices coupled to a network;

3 sensing a connection to the network of a device that is not included in the predetermined
4 configuration;
5 assigning the connected device a standby address which allows access to device information and
6 configuration parameters of the connected device;
7 commissioning the connected device into an operational state in communication with the control
8 system; and
9 configuring the connected device in combination with the predetermined configuration of devices.

1 68. A method according to Claim 67 wherein commissioning the connected device further
2 comprises:
3 assigning to the connected device a physical device tag, a device address, and a device identification;
4 installing a control strategy to the connected device; and
5 placing the connected device in an operational state in communication with the network.

1 69. A method according to Claim 67 wherein configuring the connected device further comprises:
2 interrogating the connected device to determine a device type;
3 determining a role of the connected device in the context of the predetermined configuration; and
4 assigning a physical device tag so that the determined role is set.

1 70. A computer program product comprising:
2 a computer usable medium having computable readable code embodied therein for configuring a
3 control system including:
4 a routine for predetermining a configuration of devices coupled to a network;
5 a routine for sensing a connection to the network of a device that is not included in the
6 predetermined configuration;
7 a routine for assigning the connected device a standby address which allows access to device
8 information and configuration parameters of the connected device;
9 a routine for commissioning the connected device into an operational state in communication with
10 the control system; and
11 a routine for configuring the connected device in combination with the predetermined configuration
12 of devices.

1 71. A process control system (100) for controlling a process according to a control strategy, the
2 process control system comprising:
a computer system including a processor, an input interface and a display coupled to the process;
a field device coupled to the process;
a controller coupled to the field device and communicatively coupled to the computer system; and
a software system including:

an interactive, user-directed process configuration program including a plurality of control language editors for selecting the control strategy using a control language selected from a plurality of control languages, the process configuration program creating an executable control module and downloading the executable control module selectively among the computer system, the field device, and the controller; and an executable control module selectively created, downloaded and executed, the control module being configurable by the process configuration program to configure a control language execution engine.

72. A process control system (100) comprising:

a field device;

a controller (110) coupled to the field device;

a workstation (102) coupled to the controller and having an interactive input interface and a display;

and

a software system including:

a user interface (300) responsive to the interactive input interface for interfacing the process control system to a user;

a routine for implementing a control strategy for the process control system, the control strategy being selectively apportioned into a plurality of control modules (440) and selectively distributed among the field device, the controller, and the workstation; and

an interactive, user-directed process configuration program including a plurality of control language editors for selecting the control strategy using a control language selected from a plurality of control languages, the process configuration program selectively creating the control modules implementing the control strategy, selectively apportioning the control modules, and selectively distributing the control modules among the field device, the controller, and the workstation for executing the control strategy.

73. A process control system (100) for controlling a plurality of field devices of multiple different field device types, the process control system comprising:

a plurality of controllers (110) coupled to the field devices; and

a software system including:

a user interface (300) for interfacing the process control system to a user;

a plurality of control modules (440) selectively created, downloaded, and executed on ones of the plurality of controllers, the software system for communicating with the multiple different field device types and for controlling the multiple different field devices independently of control of control modules in other field devices; and

10 an interactive, user-directed process configuration program including a plurality of control
11 language editors for selectively installing and configuring the control modules
12 using a control language selected from a plurality of control languages.

1 74. A process control system according to Claim 71, Claim 72 or Claim 73, wherein:
2 the software system is distributed among the processor and the controller including:
3 an interactive input and display routine executable on the processor, and
4 a plurality of control language execution engines executable on the controller for
5 implementing respective languages of the plurality of control languages.

1 75. A process control system according to Claim 71, Claim 72, or Claim 73 further comprising:
2 a plurality of field devices;
3 a plurality of controllers coupled to the field devices, wherein:
4 the software system is distributed among the workstation and the plurality of controllers and
5 includes:
6 an interactive input and display routine executable on the workstation, and
7 a plurality of control language execution engines selectively created, apportioned,
8 distributed, and executable on the plurality of controllers for implementing the
9 plurality of control languages.

1 76. A method for configuring a process control environment, the process control environment
2 including a computer system having a processor coupled to a display device, the method comprising:
3 providing a plurality of instructional sections, an instructional section setting forth information
4 relating to configuring the process control environment;
5 selecting a control language editor for defining a process control environment configuration;
6 activating the selected control language editor;
7 displaying, on the display device, a sequence of configuration screen presentations relating to the
8 instruction sections as directed in terms of the selected control language editor; and
9 guiding a user through the configuration of the process control environment via the sequence of
10 configuration screen presentations;
11 interactively creating a plurality of independent control modules (440);
12 configuring the plurality of control modules to create a control strategy;
13 transferring the control strategy to a selected controller executing a control language execution
14 engine corresponding to the selected control language editor, the selected controller
15 executing the control strategy independently of execution of other control strategies.

1 77. A computer program product comprising:

2 a computer usable medium having computable readable code embodied therein for controlling a
3 process control system (100) according to a control strategy, the process control system
4 including a computer system with a processor, an input interface and a display coupled to
5 the process; a field device coupled to the process; a controller coupled to the field device
6 and communicatively coupled to the computer system; and a software system further
7 including:
8 an interactive, user-directed process configuration program including a plurality of control
9 language editors for selecting the control strategy using a control language selected
10 from a plurality of control languages, the process configuration program creating
11 an executable control module and downloading the executable control module
12 selectively among the computer system, the field device, and the controller; and
13 an executable control module selectively created, downloaded and executed, the control
14 module being configurable by the process configuration program to configure a
15 control language execution engine.

1 78. A computer program product comprising:

2 a computer usable medium having computable readable code embodied therein for managing a
3 process control system (100) including a field device, a controller coupled to the field
4 device, a workstation (102) coupled to the controller and having an interactive input
5 interface and a display; and a software system including:
6 a user interface (300) responsive to the interactive input interface for interfacing the process
7 control system to a user;
8 a routine for implementing a control strategy for the process control system, the control
9 strategy being selectively apportioned into a plurality of control modules (440)
10 and selectively distributed among the field device, the controller, and the
11 workstation; and
12 an interactive, user-directed process configuration program including a plurality of control
13 language editors for selecting the control strategy using a control language selected
14 from a plurality of control languages, the process configuration program
15 selectively creating the control modules implementing the control strategy,
16 selectively apportioning the control modules, and selectively distributing the
17 control modules among the field device, the controller, and the workstation for
18 executing the control strategy.

1 79. A computer program product comprising:

2 a computer usable medium having computable readable code embodied therein for controlling a
3 plurality of field devices of multiple different field device types, the process control system

(100) including a plurality of controllers (110) coupled to the field devices and a software system including:
a user interface (300) for interfacing the process control system to a user;
a plurality of control modules (440) selectively created, downloaded, and executed on ones of the plurality of controllers, the software system for communicating with the multiple different field device types and for controlling the multiple different field devices independently of control of control modules in other field devices; and
an interactive, user-directed process configuration program including a plurality of control language editors for selectively installing and configuring the control modules using a control language selected from a plurality of control languages.

80. A computer program product comprising:

a computer usable medium having computable readable code embodied therein for configuring a process control environment, the process control environment including a computer system having a processor coupled to a display device and a software system including:
a routine for providing a plurality of instructional sections, an instructional section setting forth information relating to configuring the process control environment;
a routine for selecting a control language editor for defining a process control environment configuration;
a routine for activating the selected control language editor;
a routine for displaying, on the display device, a sequence of configuration screen presentations relating to the instruction sections as directed in terms of the selected control language editor; and
a routine for guiding a user through the configuration of the process control environment via the sequence of configuration screen presentations;
a routine for interactively creating a plurality of independent control modules (440);
a routine for configuring the plurality of control modules to create a control strategy;
a routine for transferring the control strategy to a selected controller executing a control language execution engine corresponding to the selected control language editor, the selected controller executing the control strategy independently of execution of other control strategies.

81. A process control system (100) comprising:

a field device including a source of event condition information;
a controller (110) coupled to the field device;
a workstation (102) coupled to the controller and including a user interface (300); and
a software system implementing an event condition monitoring and display program for the process control system, the event condition monitoring and display program including:

7 a plurality of control modules (440) including event attributes, the control modules being
8 selectively controlled and selectively distributed among the field device, the
9 controller and the workstation, the control modules operating mutually
10 independently and in parallel accumulating event condition information;
11 a display routine for accessing the event condition information from the plurality of control
12 modules and displaying the event condition information accessed from the
13 plurality of control modules in an order of priority selected by a user; and
14 a configuration routine for user-selectively defining and creating the control modules and
15 the event attributes of the control modules, and for user selectively distributing the
16 control modules among the field device, the controller, and the workstation.

1 82. A process control system according to claim 81, wherein the software system further comprises:
2 a routine for priming the event attributes with a current alarm set subject to a current user
3 responsibility scope to begin accumulating an event count when a new user logs-on.

1 83. A method for operating a process control system (100) comprising the steps of:
2 defining a plurality of conditions as events, the events being associated with a plant area;
3 activating an event journal for logging event conditions;
4 configuring an alarm behavior including the steps of:
5 creating an alarm attribute, the alarm attribute being selectively created in a control module
6 or in an equipment module;
7 selectively disabling or enabling the created alarm attribute;
8 selectively placing an enabled alarm attribute in an acknowledged state or an
9 unacknowledged state; and
10 selectively placing the alarm attribute in an active state or an inactive state;
11 selecting a priority for the alarm attribute; and
12 consolidating a plurality of active alarm conditions into a subset of highest priority alarms.

1 84. A computer program product comprising:
2 a computer usable medium having computable readable code embodied therein including a process
3 control system (100) including a field device including a source of event condition
4 information, a controller coupled to the field device, and computer coupled to the controller
5 and including a user interface (300), the computer program product including an event
6 condition monitoring and display program for the process control system, the event
7 condition monitoring and display program including:
8 a plurality of control modules (440) including event attributes, the plurality of control
9 modules being selectively controlled and selectively distributed among the field

10 device, the controller and the workstation, the control modules operating mutually
11 independently and in parallel accumulating event condition information;
12 a display routine for accessing the event condition information from the plurality of control
13 modules and displaying the event condition information accessed from the
14 plurality of control modules in an order of priority selected by the user; and
15 a configuration routine for user-selectively defining and creating the control modules and
16 the event attributes of the control modules, and for user selectively distributing the
17 control modules among the field device, the controller, and the workstation.

1 85. A computer program product comprising:

2 a computer usable medium having computable readable code embodied therein including a process
3 control system (100) further including:

4 a routine for defining a plurality of conditions as events, the events being associated with a
5 plant area;

6 a routine for activating an event journal for logging event conditions;

7 a routine for configuring an alarm behavior including:

8 a routine for creating an alarm attribute, the alarm attribute being selectively
9 created in a control module or in an equipment module;

10 a routine for selectively disabling or enabling the created alarm attribute;

11 a routine for selectively placing an enabled alarm attribute in an acknowledged
12 state or an unacknowledged state; and

13 a routine for selectively placing the alarm attribute in an active state or an inactive
14 state;

15 a routine for selecting a priority for the alarm attribute; and

16 a routine for consolidating a plurality of active alarm conditions into a subset of
17 highest priority alarms.

1 86. A computer program product according to either Claim 84 or Claim 85 further comprising:

2 a routine for priming the event attributes with a current alarm set subject to a current user
3 responsibility scope to begin accumulating an event count when a new user logs-on.

1 87. A computer program product according to either Claim 84 or Claim 85, further comprising:

2 a routine for transitioning between a plurality of alarm attribute states, including:

3 a routine for selectively disabling or enabling an alarm attribute, the alarm attribute being
4 in an inactive and acknowledged condition when the alarm attribute is disabled;

5 a routine for configuring the alarm attribute with a boolean attribute so that an alarm is
6 active when the boolean attribute is true and the alarm is inactive when the
7 boolean attribute is false;

8 a routine for optionally inverting the sense of the boolean attribute with an invert attribute;
9 and
10 a routine for when the alarm attribute is enabled, selectively acknowledging the alarm
11 attribute or unacknowledging the alarm attribute.

1 88. A computer program product according to either Claim 84 or Claim 85, wherein the routine for
2 consolidating a plurality of active alarm conditions includes:

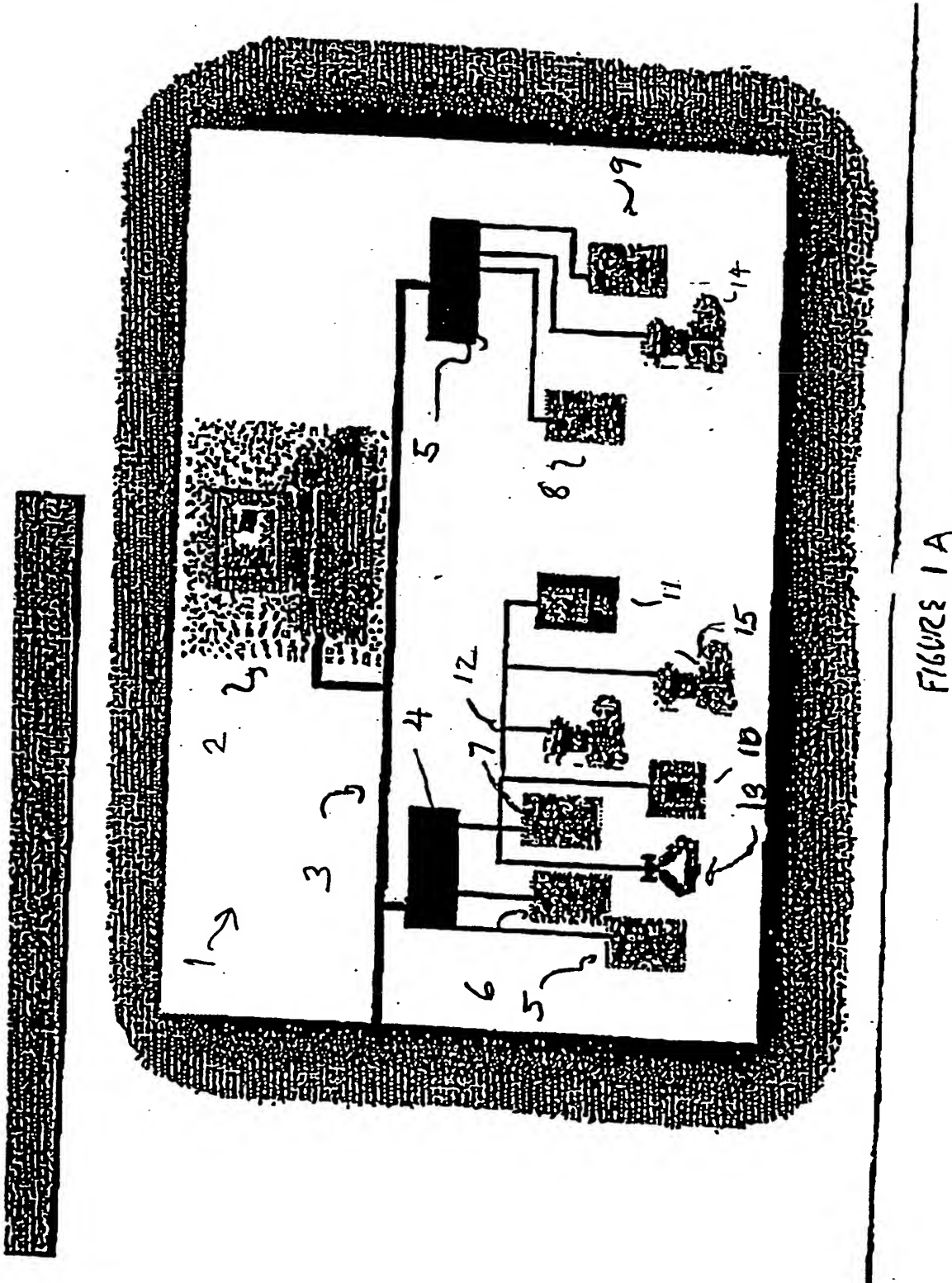
3 a routine for ranking an alarm condition in an order of decreasing order of precedence based on an
4 ordered priority including:
5 a routine for first ranking an unacknowledged condition with precedence over an
6 acknowledged condition;
7 a routine for second ranking a condition in order of High selected priority, Medium selected
8 priority, then Low selected priority; and
9 a routine for third ranking a condition in order of time of detection with a newest
10 timestamp condition having a precedence over an older timestamp condition.

1 89. A computer program product according to either Claim 84 or Claim 85, wherein the routine for
2 consolidating a plurality of active alarm conditions includes:

3 a routine for user-level consolidation including:
4 a routine for displaying an alarm banner in a graphical user interface (300);
5 a routine for granting alarm privilege over a one or more plant areas to a user;
6 a routine for predefining an attribute container supporting an alarms attribute;
7 a routine for constructing a display referencing the attribute container; and
8 a routine for allowing access to highest priority alarms based on the display referencing the
9 attribute container.

1 90. A computer program product according to either Claim 84 or Claim 85, wherein:
2 the conditions defined as events are conditions selected from among the conditions of:

3 alarms;
4 alarm acknowledgments;
5 user changes including attributes written by the user, methods invoked by the user, and log-
6 in and log-out operations by the user;
7 configuration changes to a run time system including system installation and de-installation
8 operations;
9 Sequential Function Chart (SFC) state changes;
10 Operator Attention Requests (OARs); and
11 miscellaneous events including non-alarm state transitions and equipment state changes.



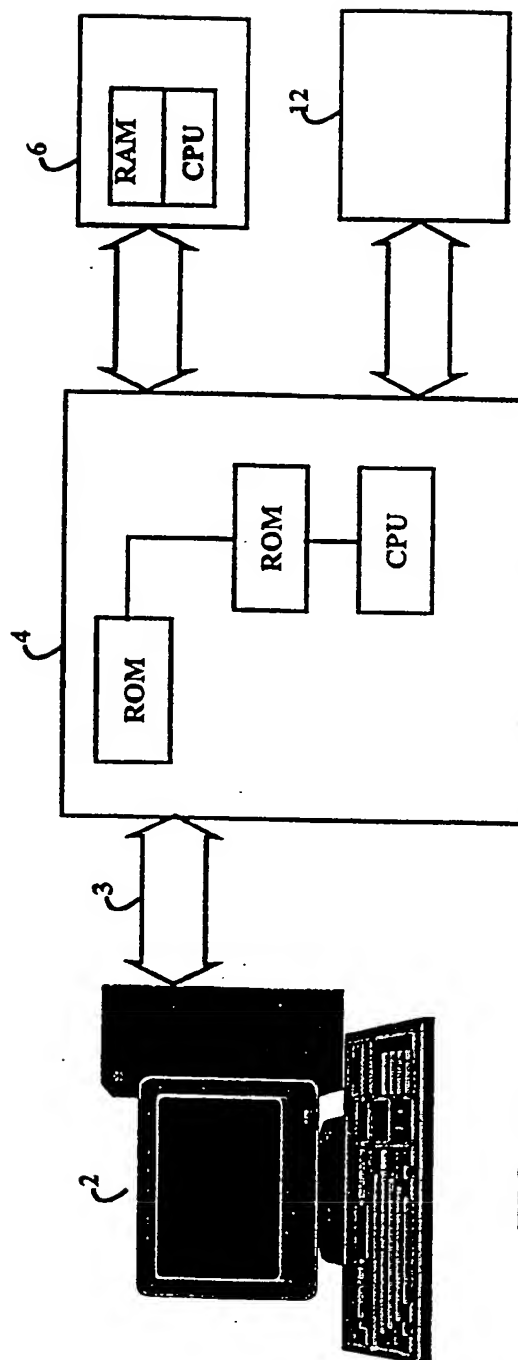
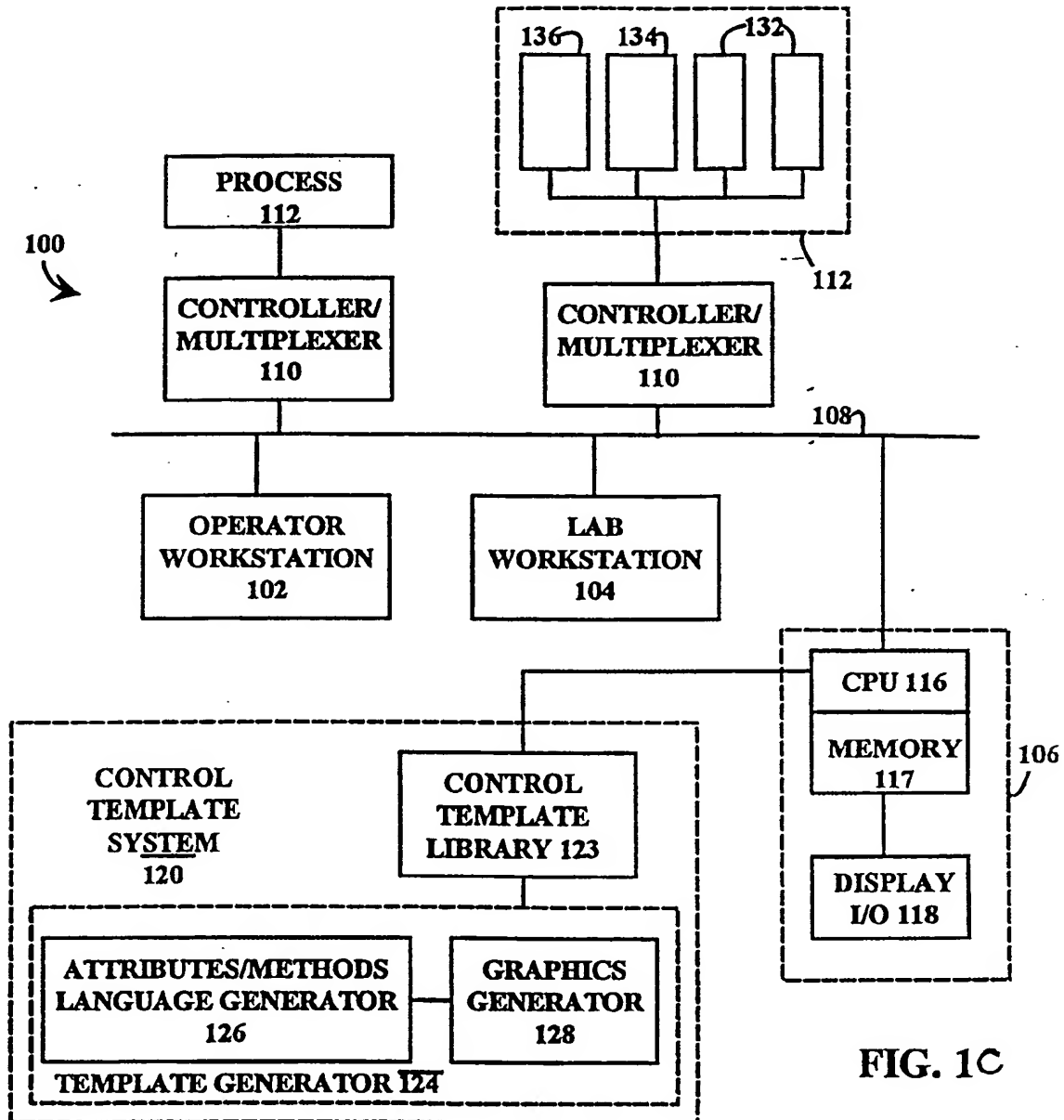
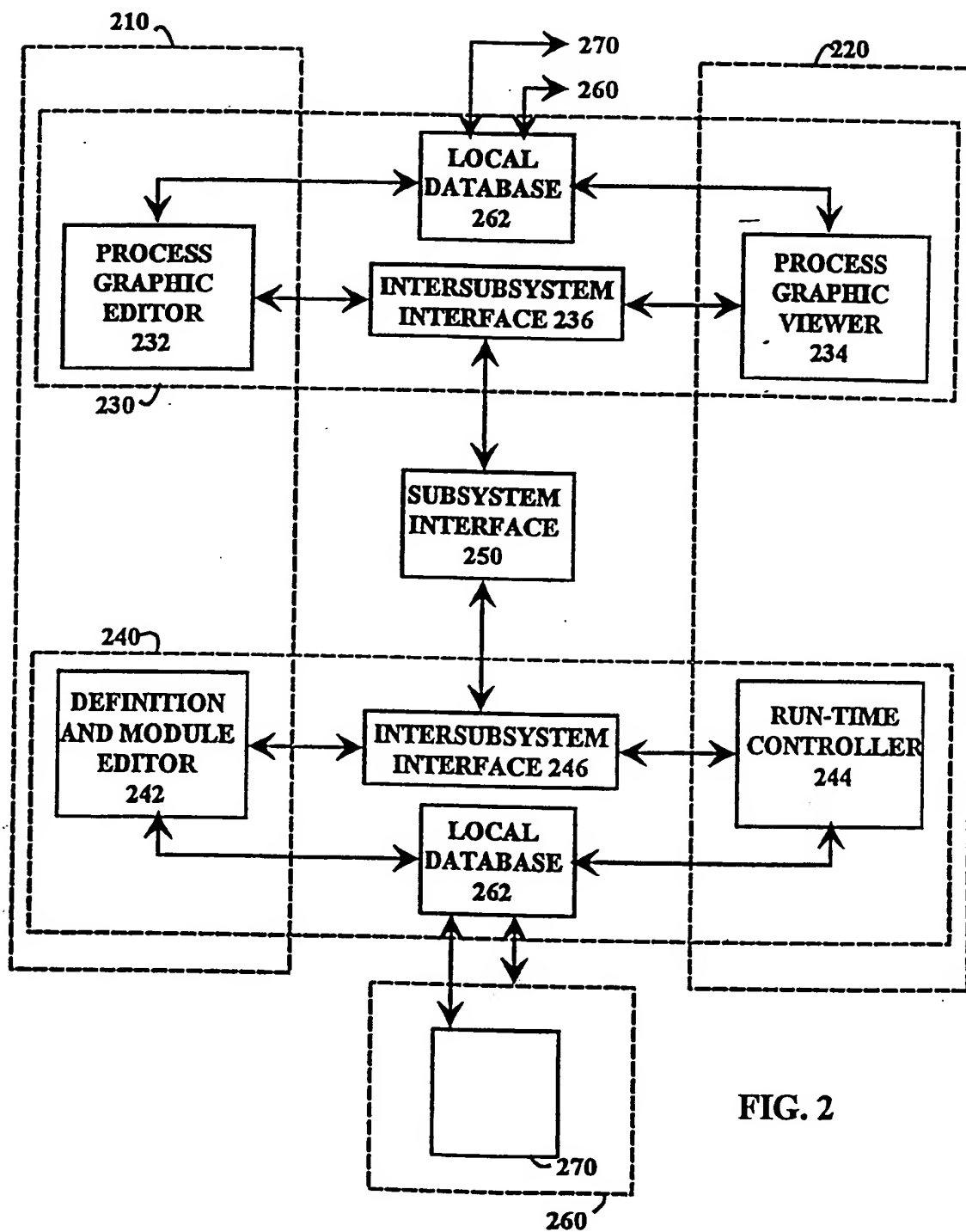


FIG. 1B





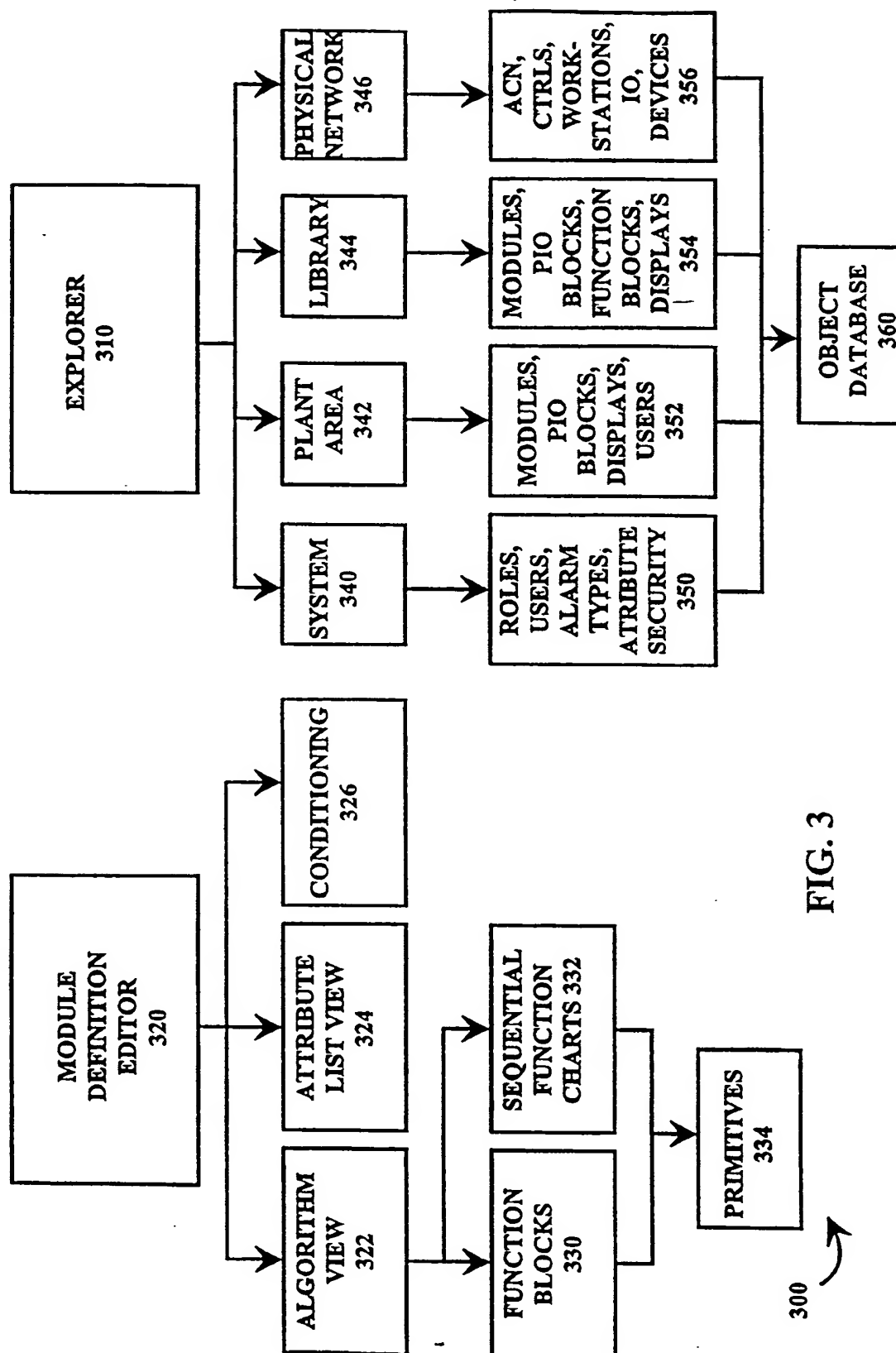


FIG. 3

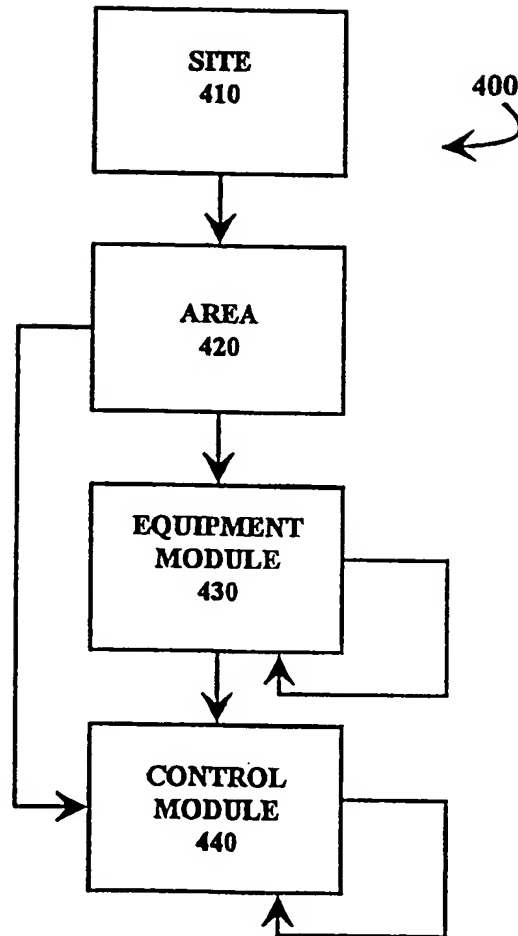


FIG. 4

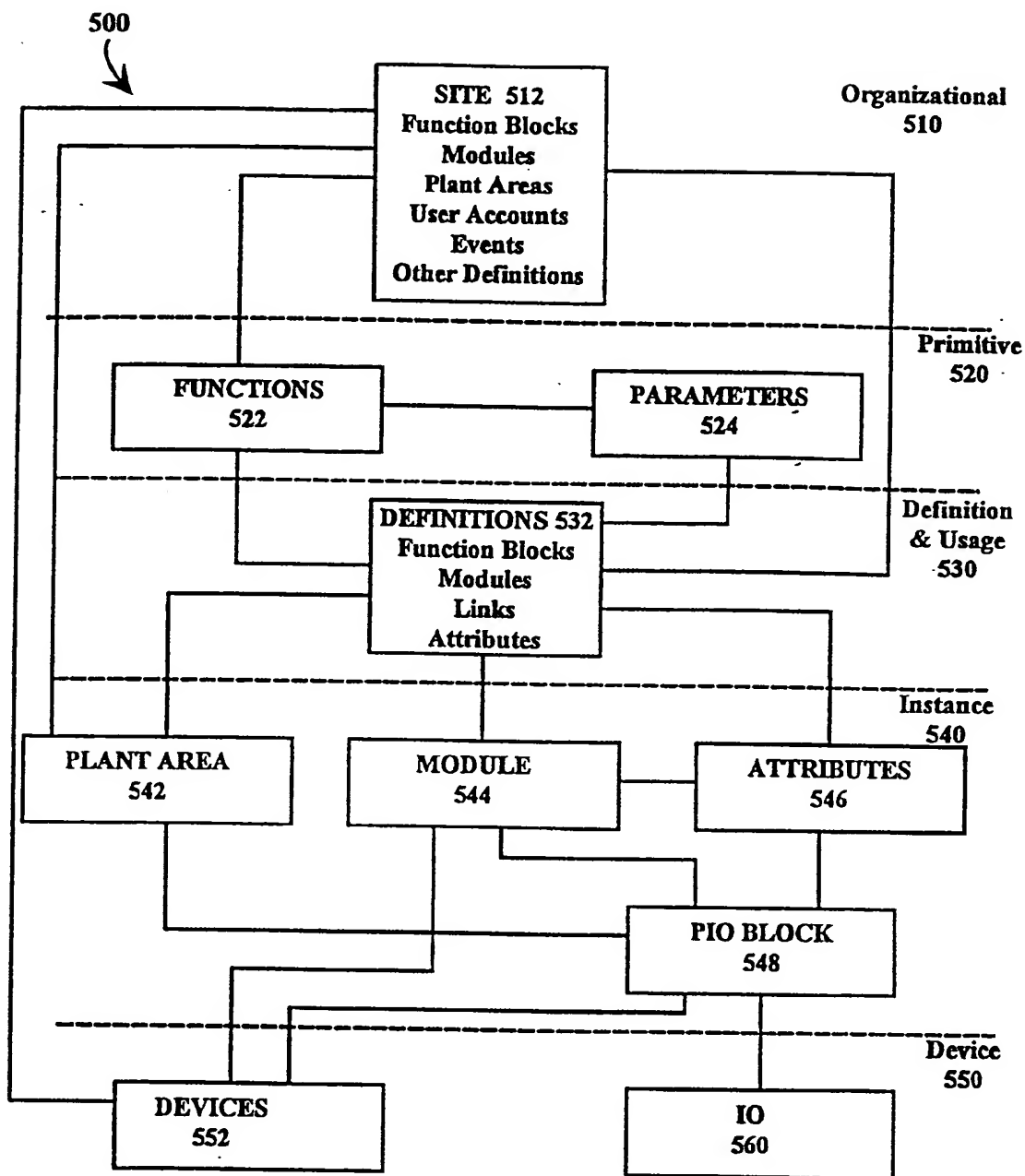


FIG. 5

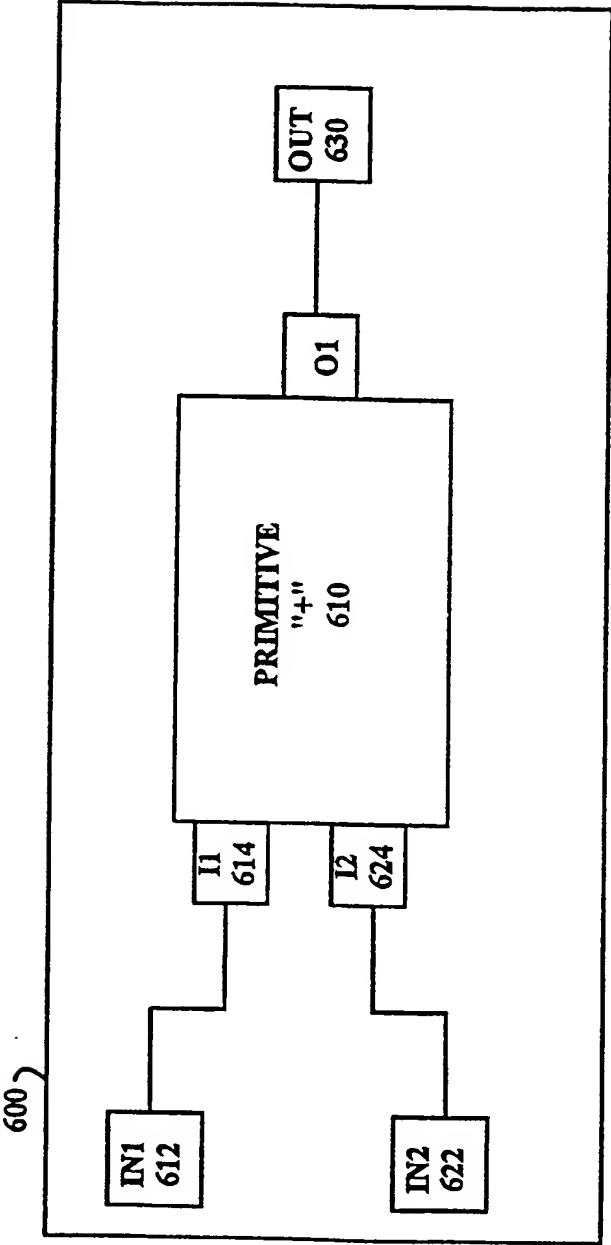


FIG. 6

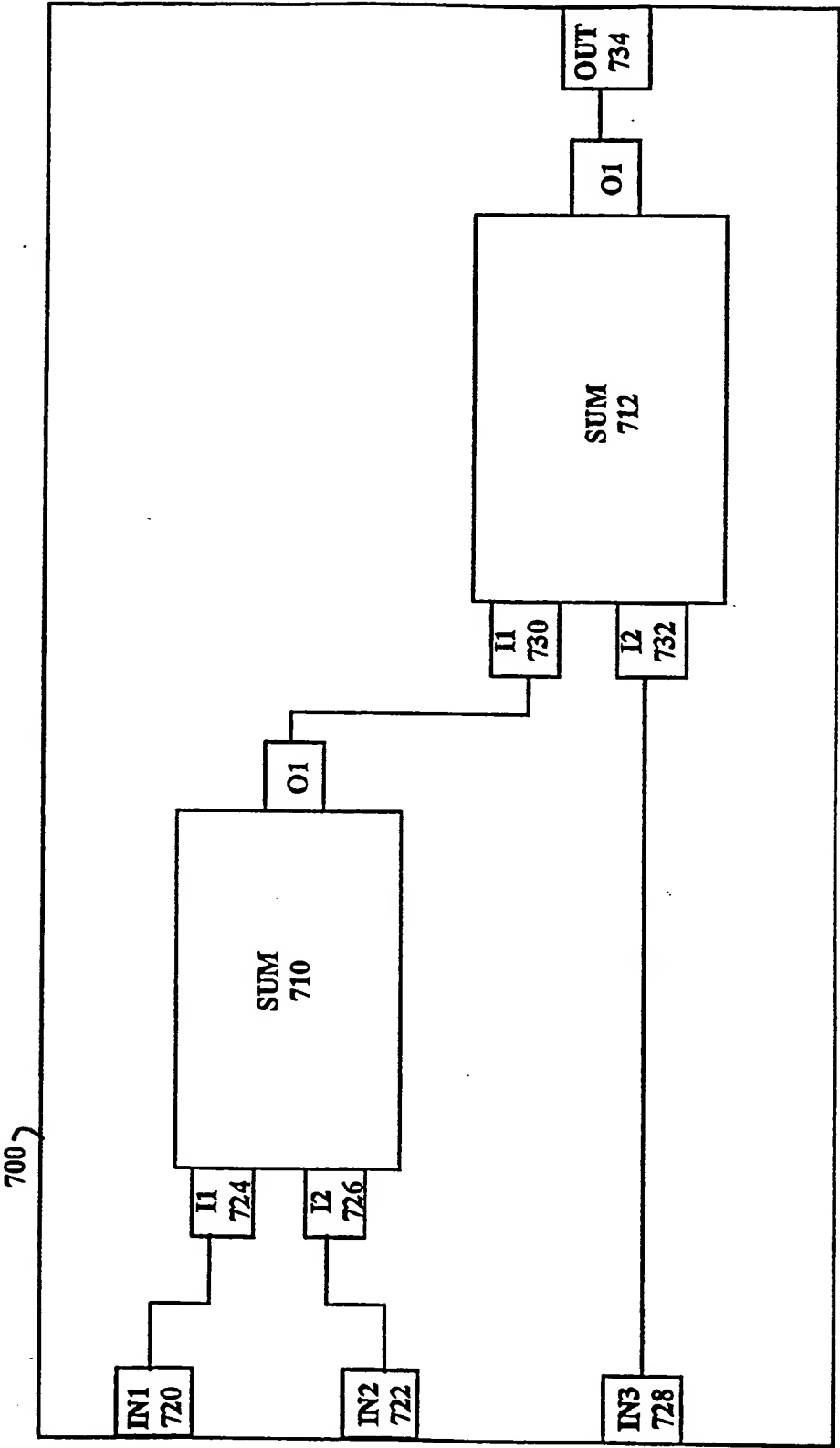


FIG. 7

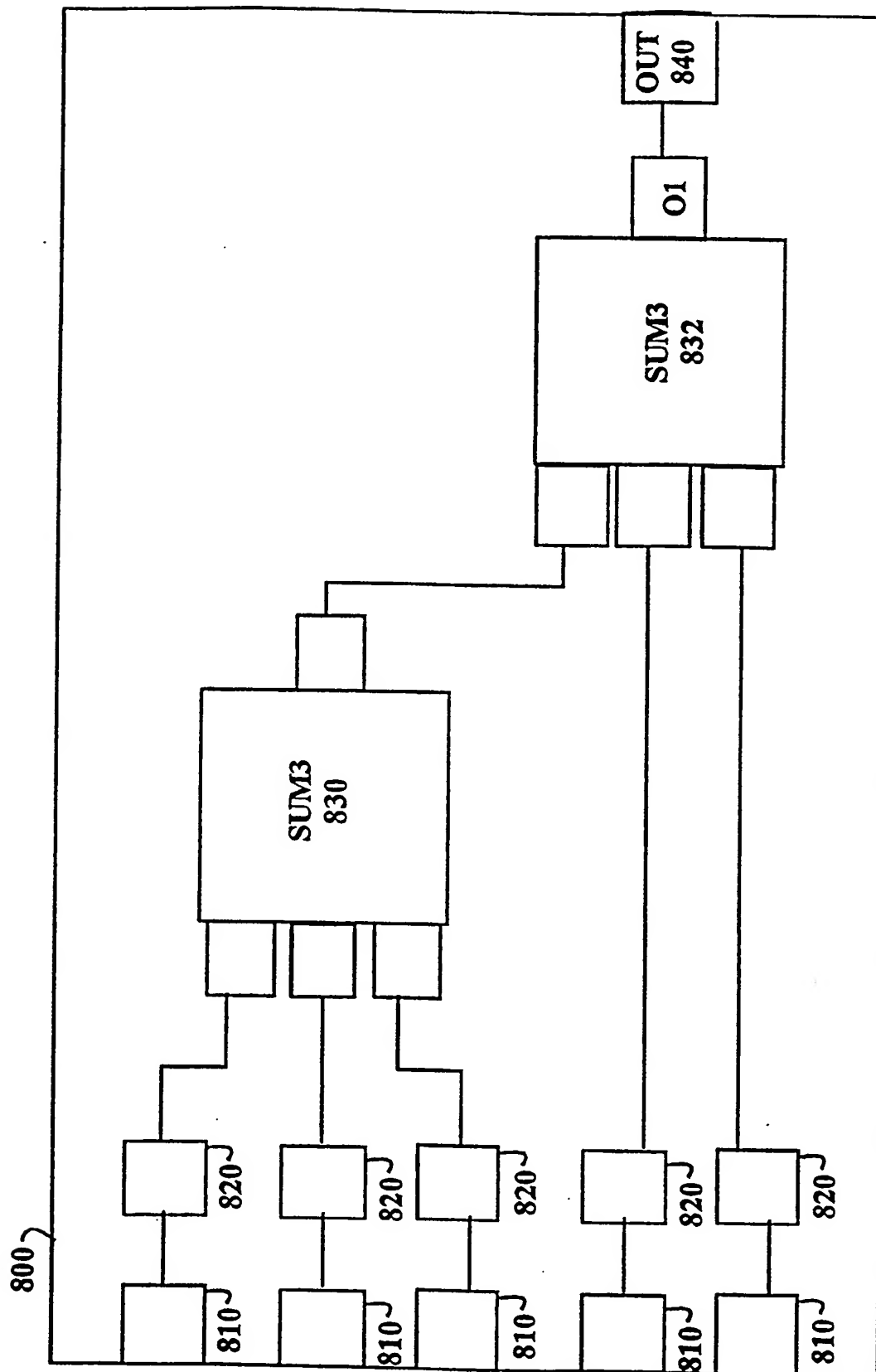


FIG. 8

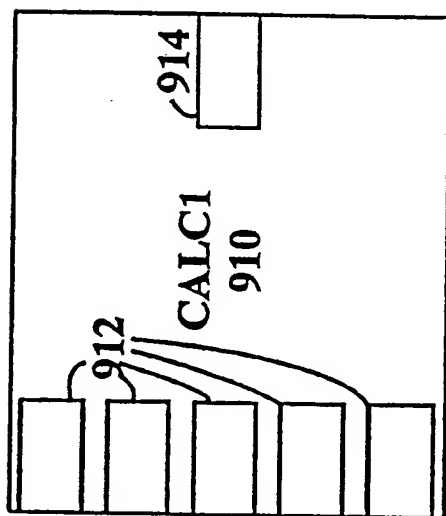
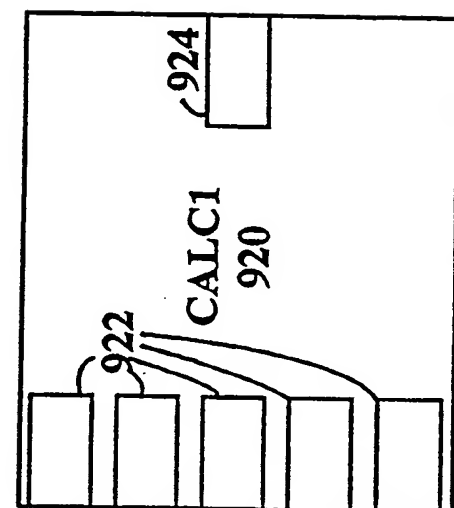


FIG. 9

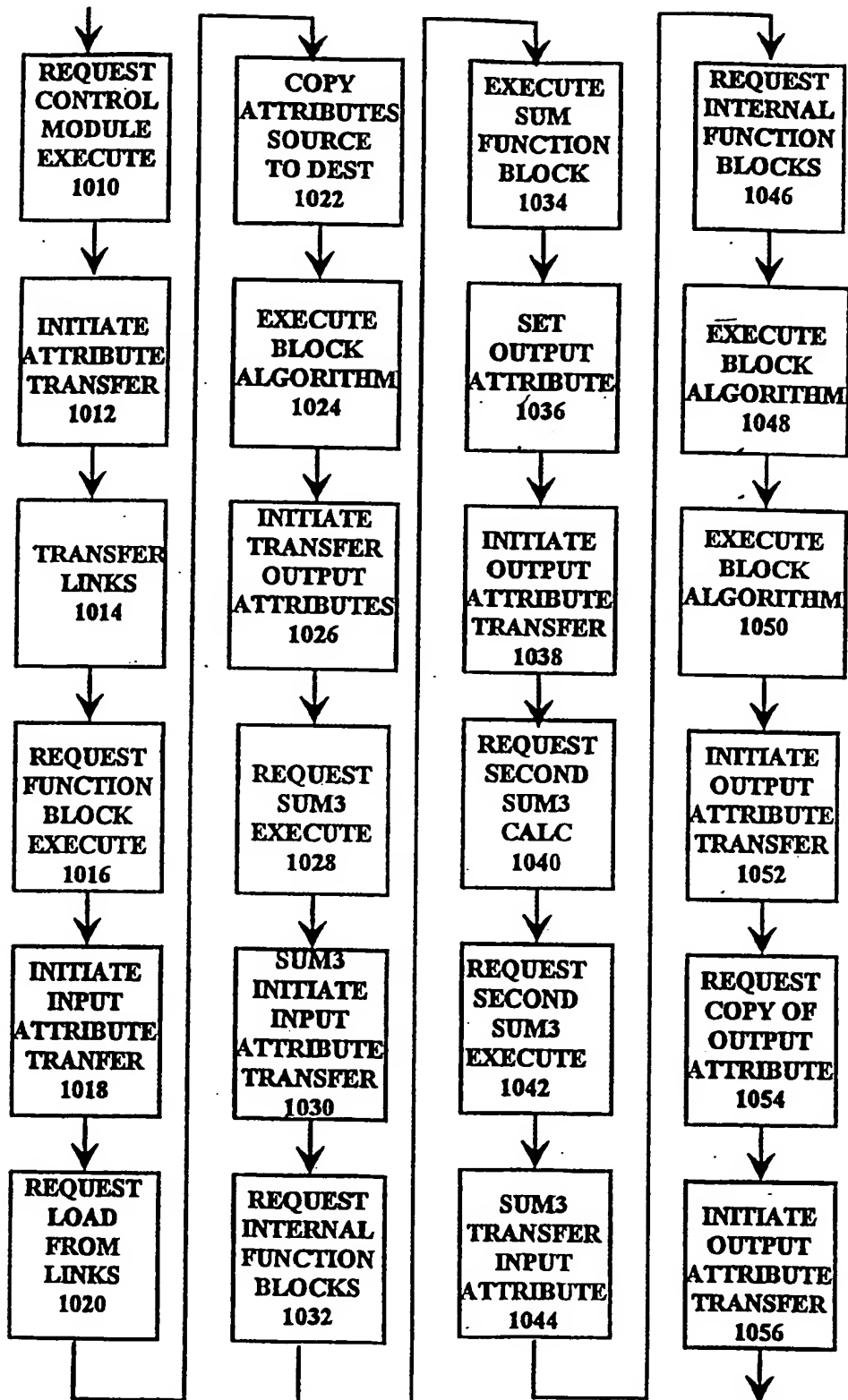


FIG. 10

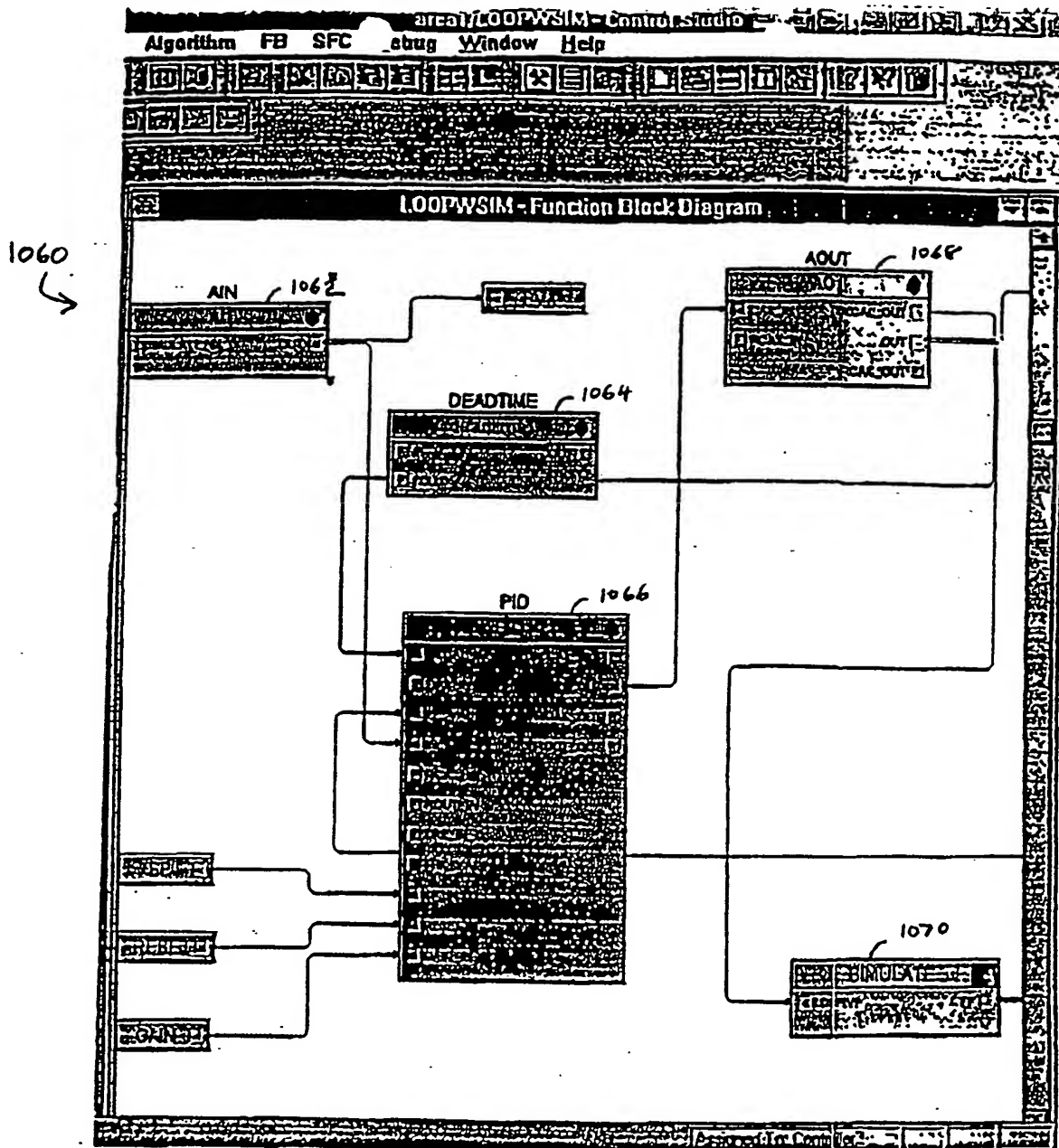
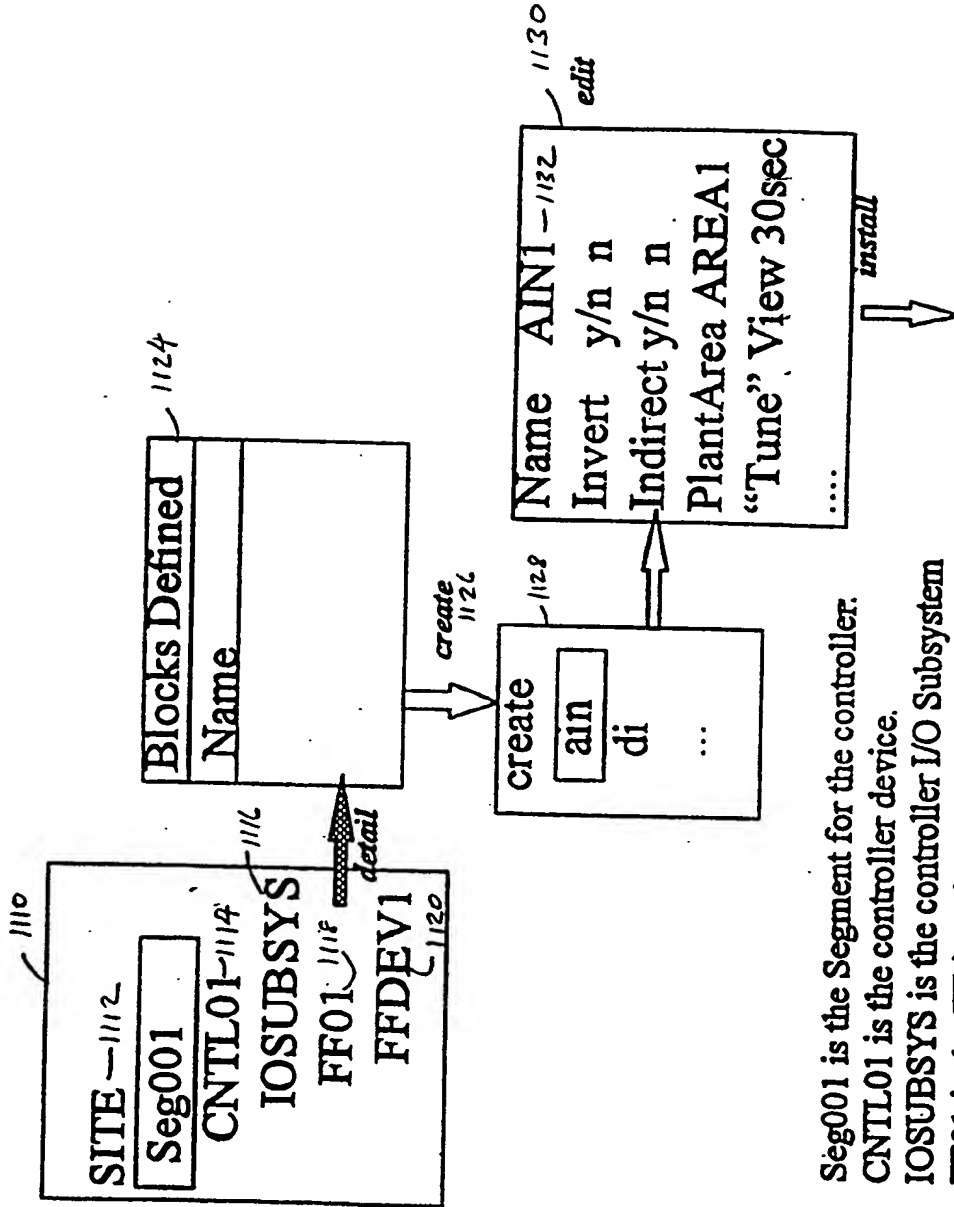


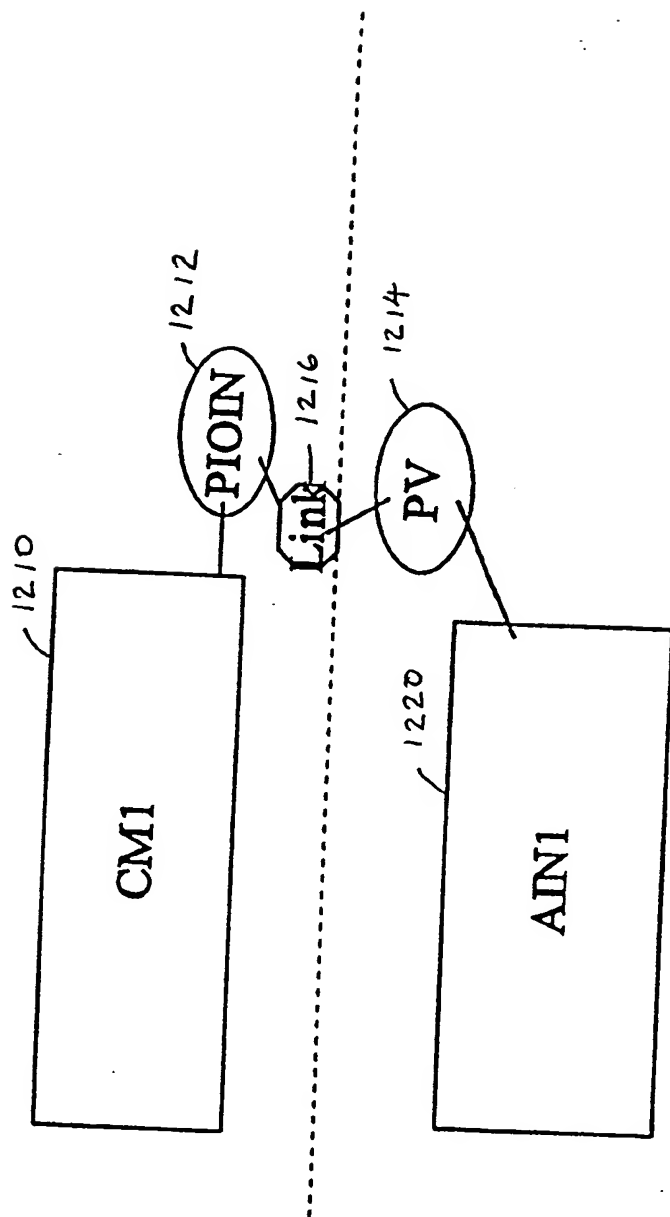
FIG. 11

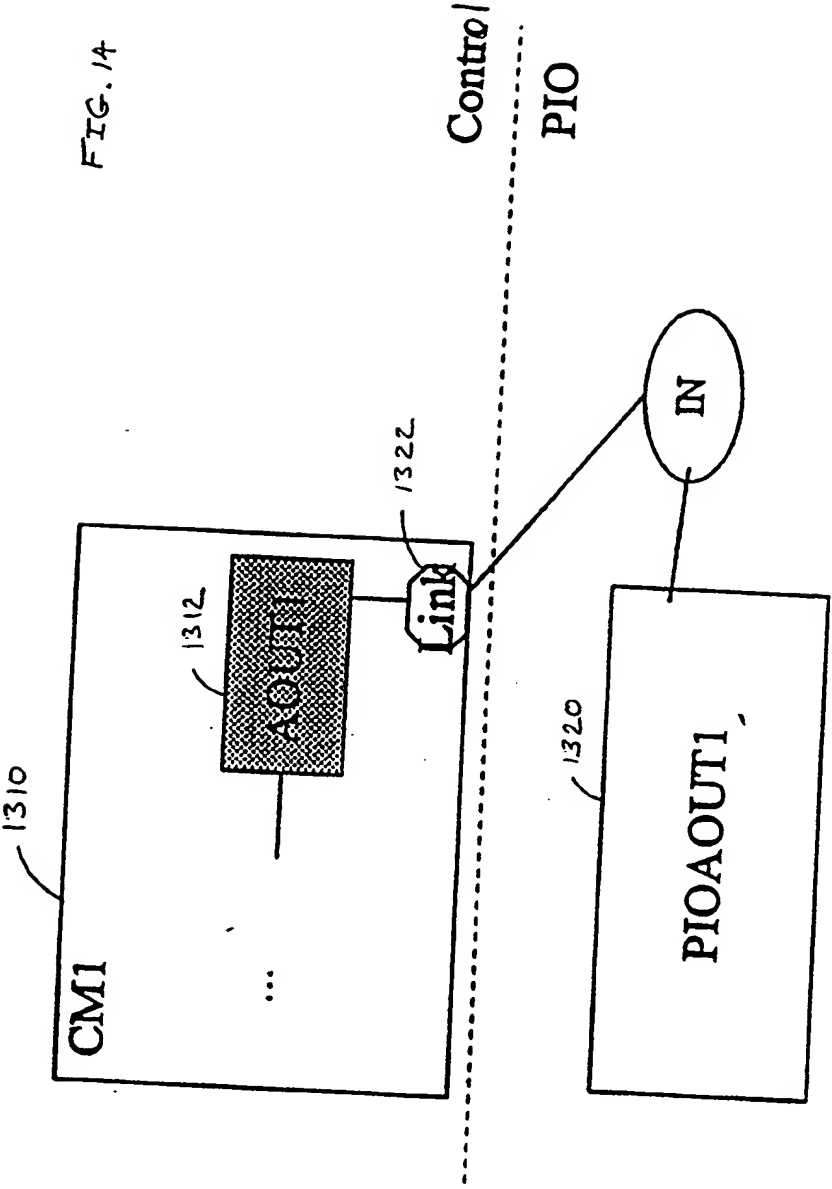
FIG. 12



Seg001 is the Segment for the controller.
 CNTL01 is the controller device.
 IOSUBSYS is the controller I/O Subsystem
 FF01 is the FF interface device.
 FFDEV1 is the actual FF device.

FIG. 13





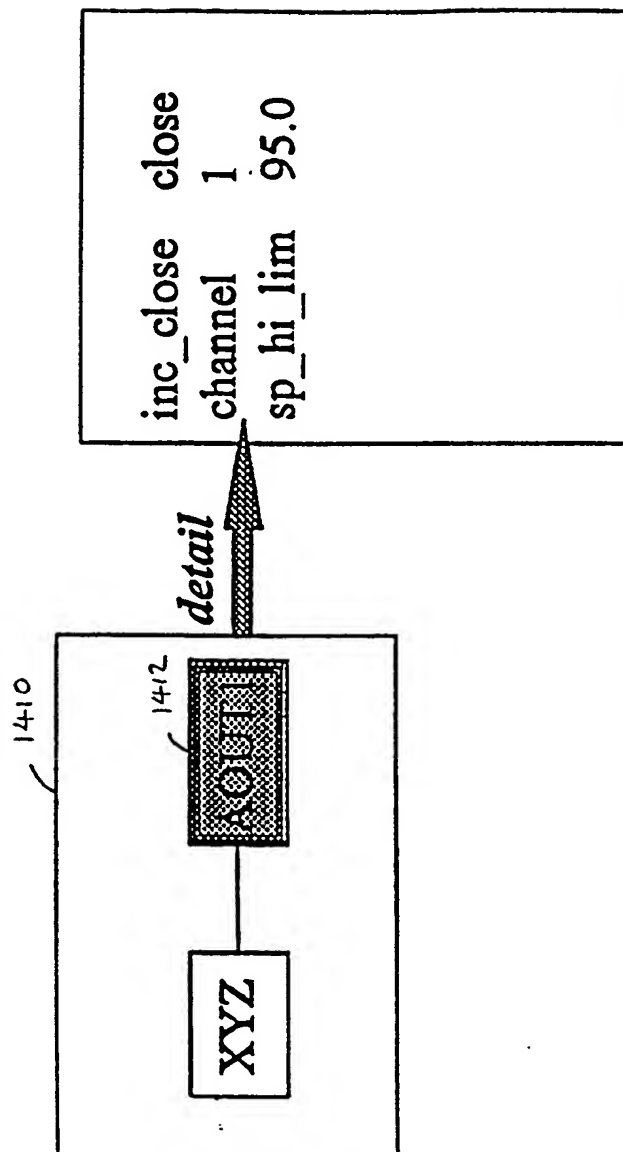


FIGURE 15

System IST Table

1510

| STName | Node | IOsys | Card | Port | Dev | Blk | Signal | Attr | St. Type | Unit |
|--------|------|-------|------|------|-----|-----|-----------|------|----------|------|
| FI101 | CON1 | 1 | 1 | 1 | | | FIELD_VAL | AIN | | |
| FI102 | CON1 | 1 | 2 | 1 | 1 | | FIELD_VAL | AIN | | |

(Used to address the Device,
IO subsystem Card, Port, Device
and Block property data.)

AIN Signal Table

1520

| BU100 | BU0 | ETC |
|-------|-----|-----|
| 100 | 0 | ... |
| 1000 | 100 | ... |
| 1000 | 100 | ... |

FIG. 16

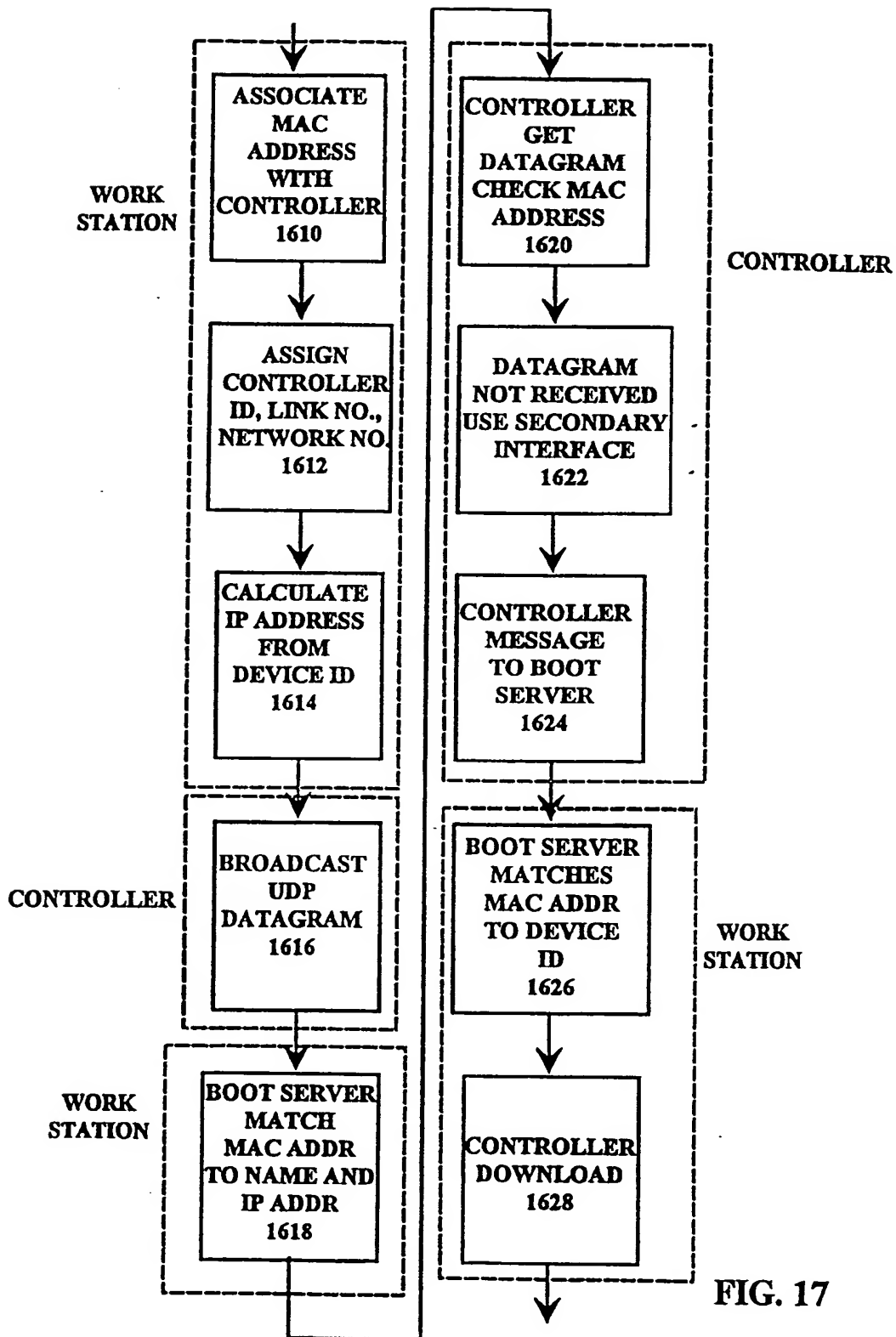


FIG. 17

FIG. 18

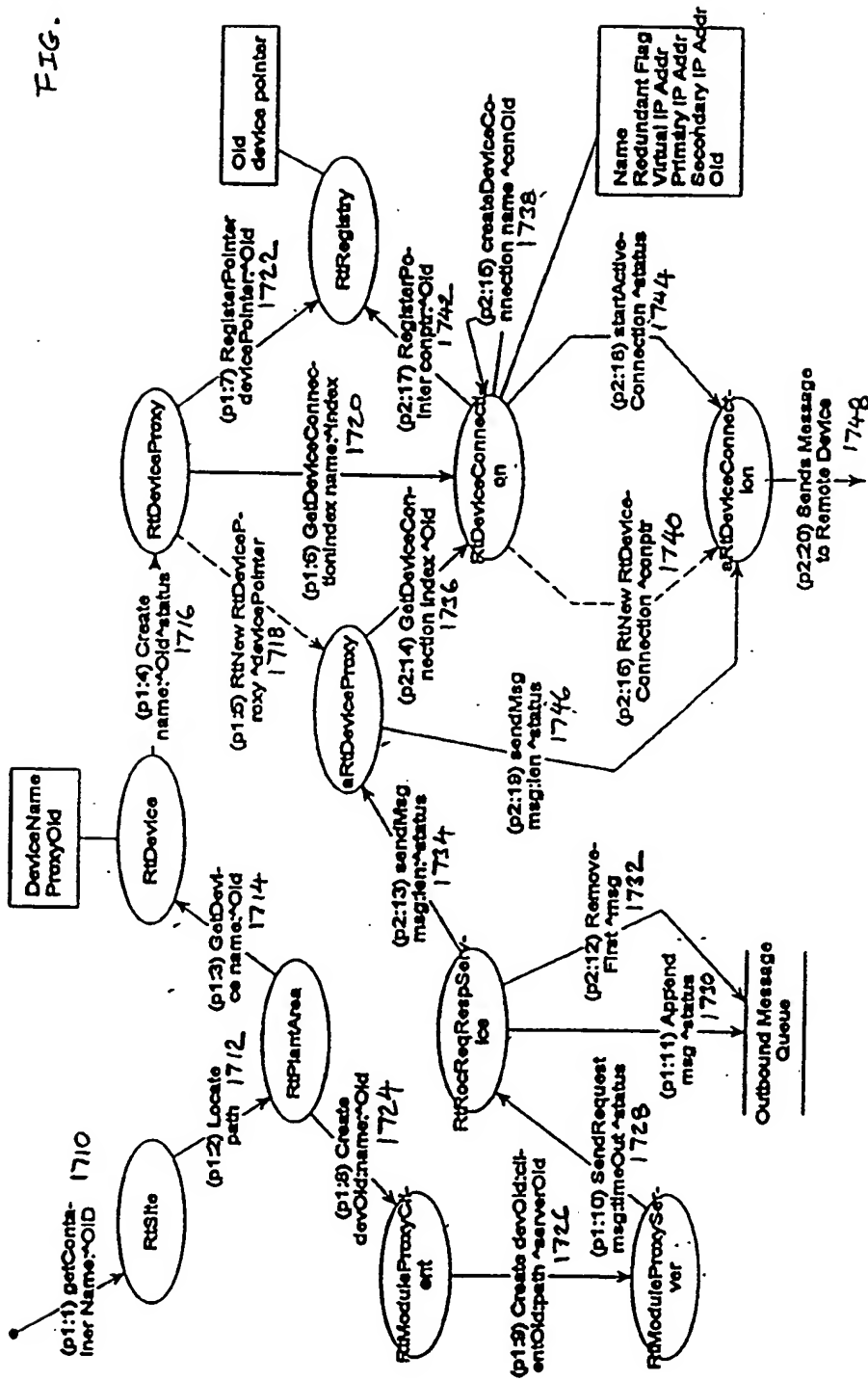


FIG. 19

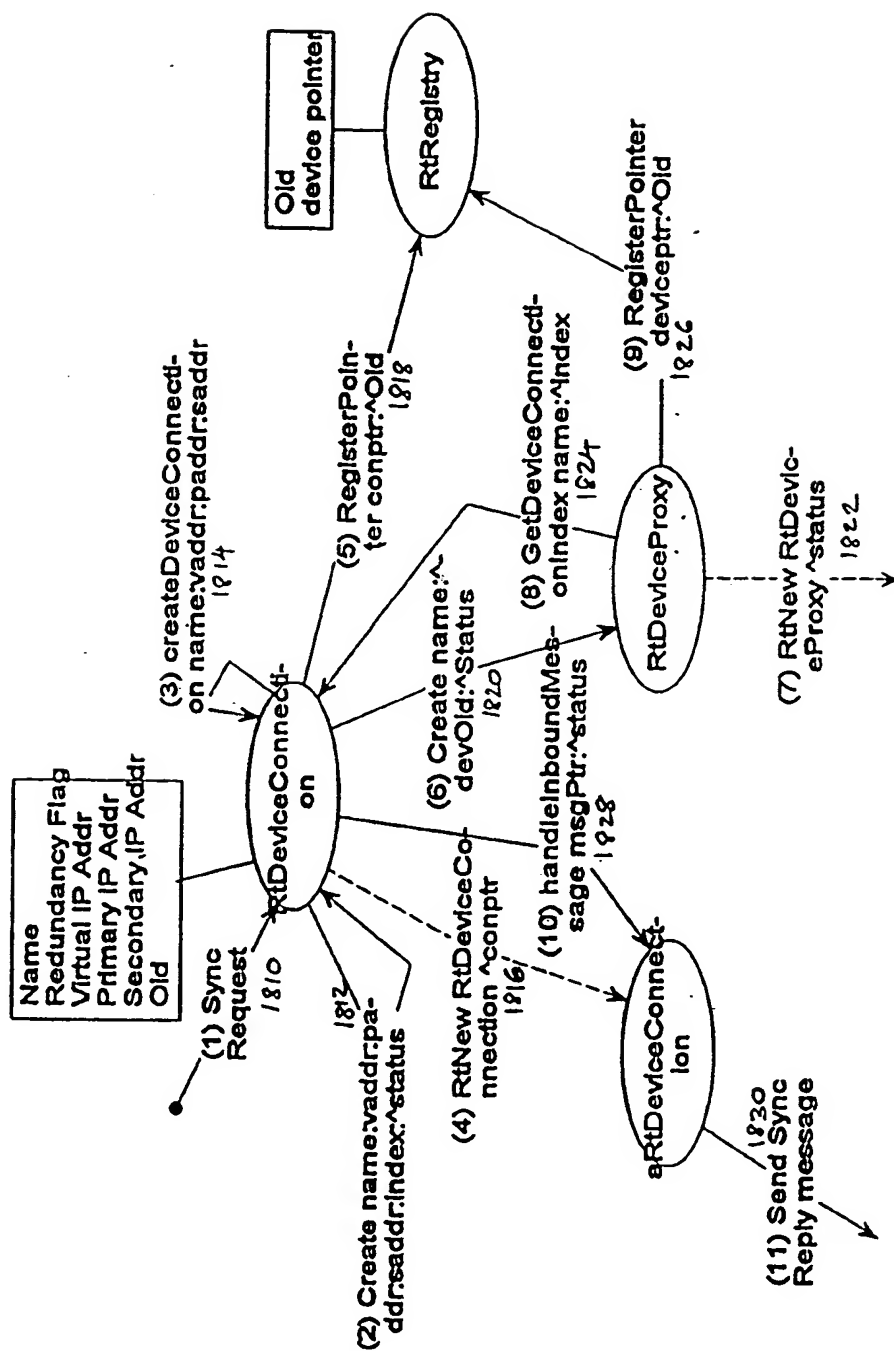


FIG. 20

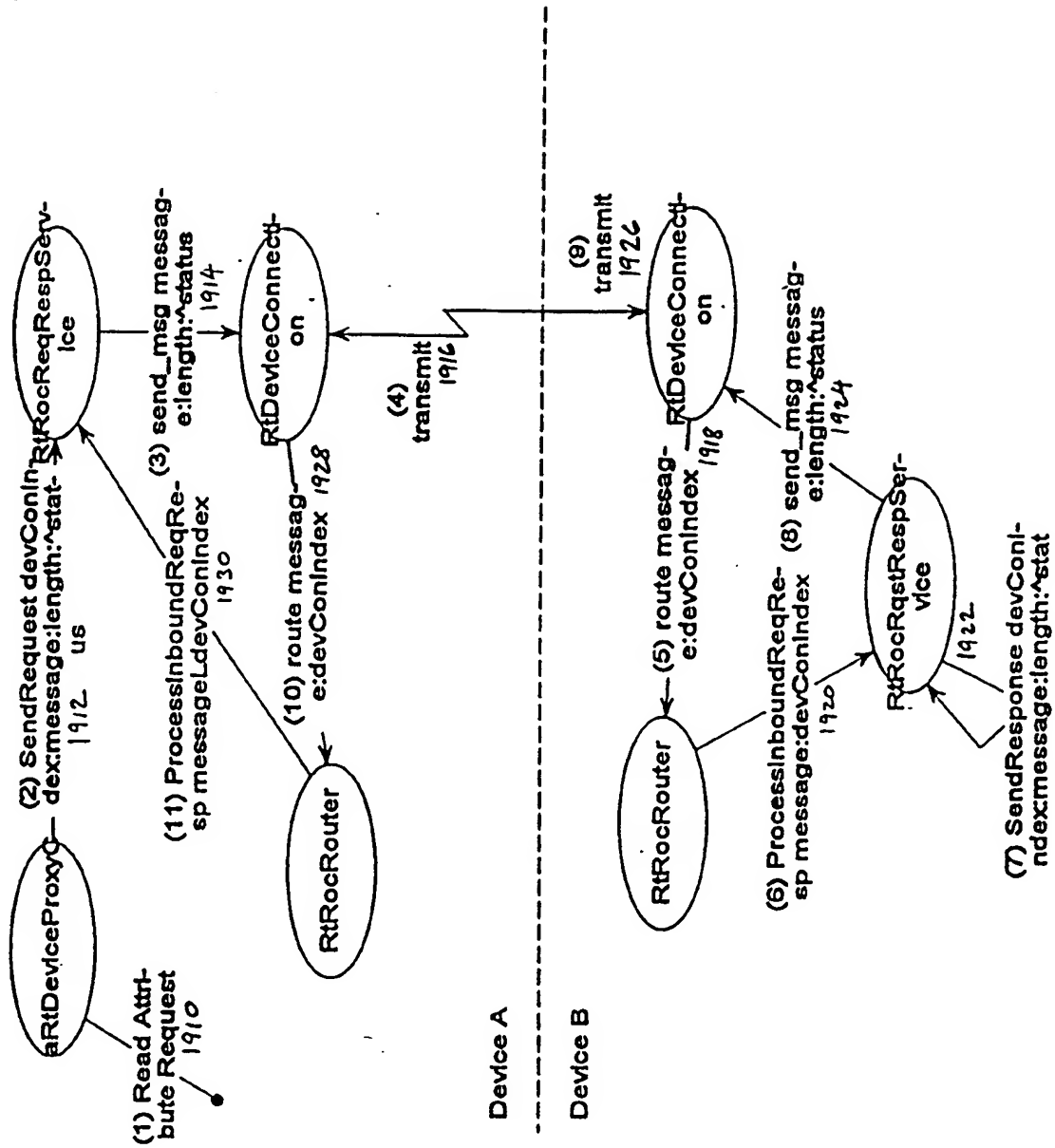


FIG. 21

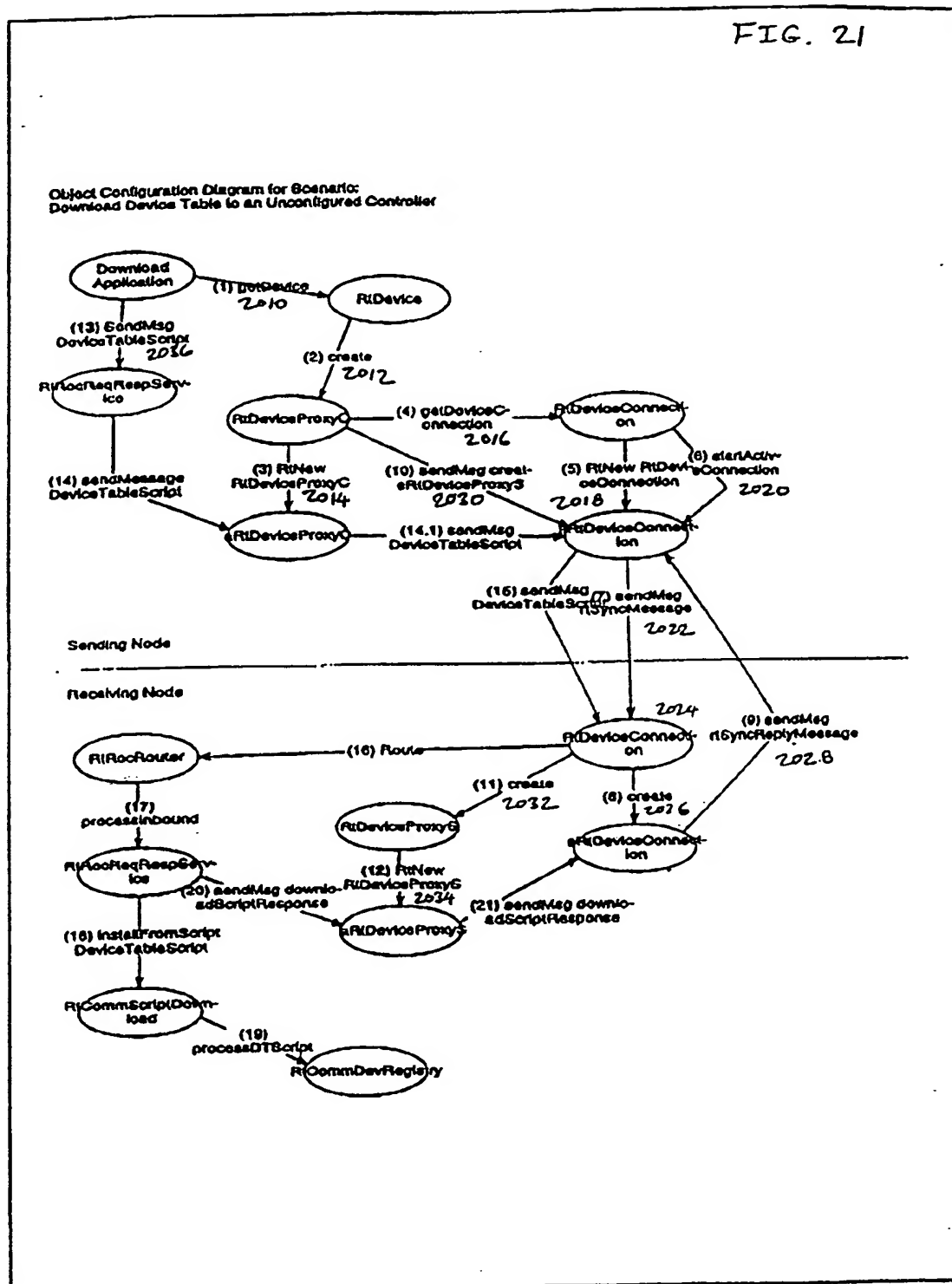


FIG. 22

2200

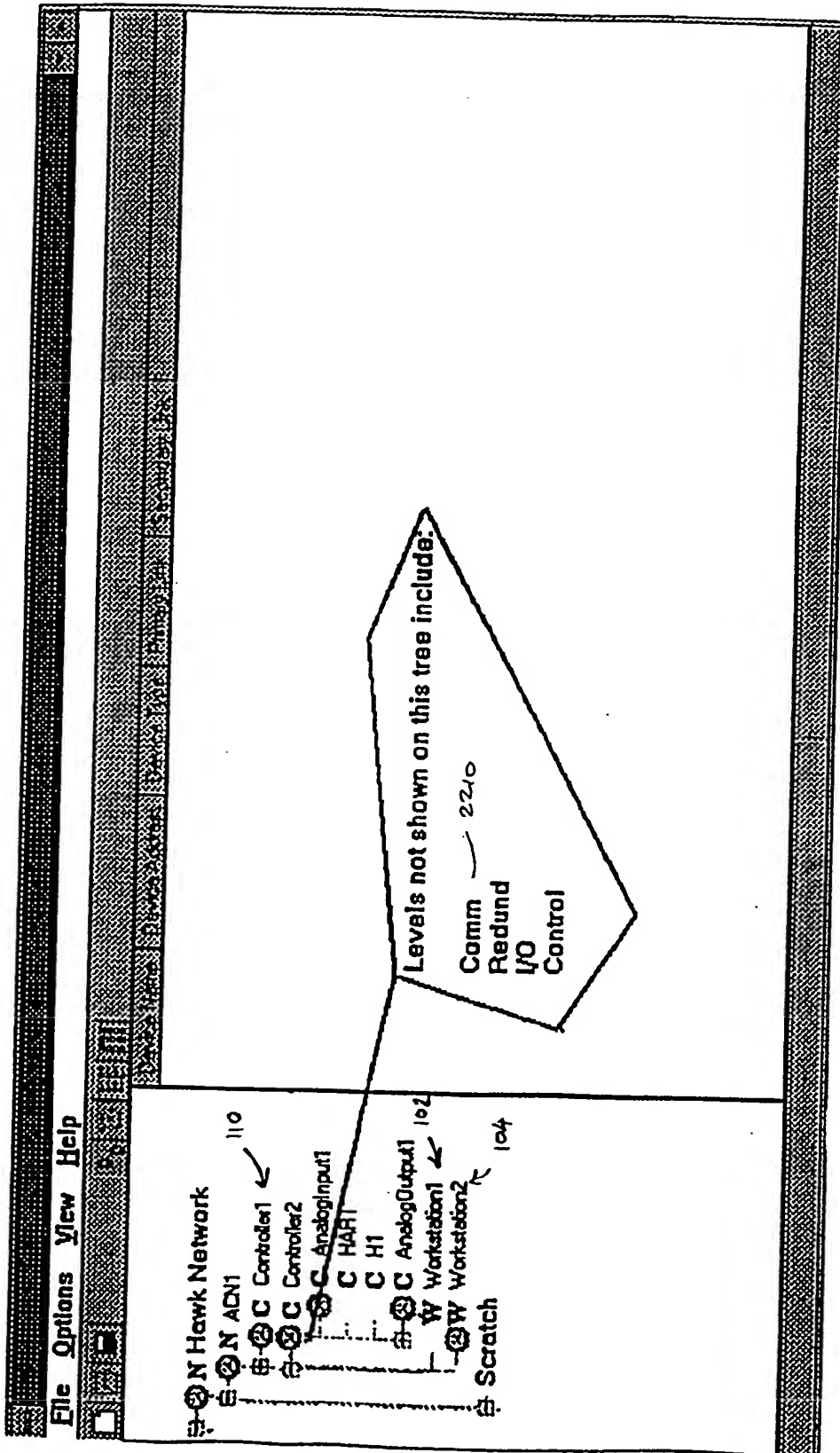


FIG. 23

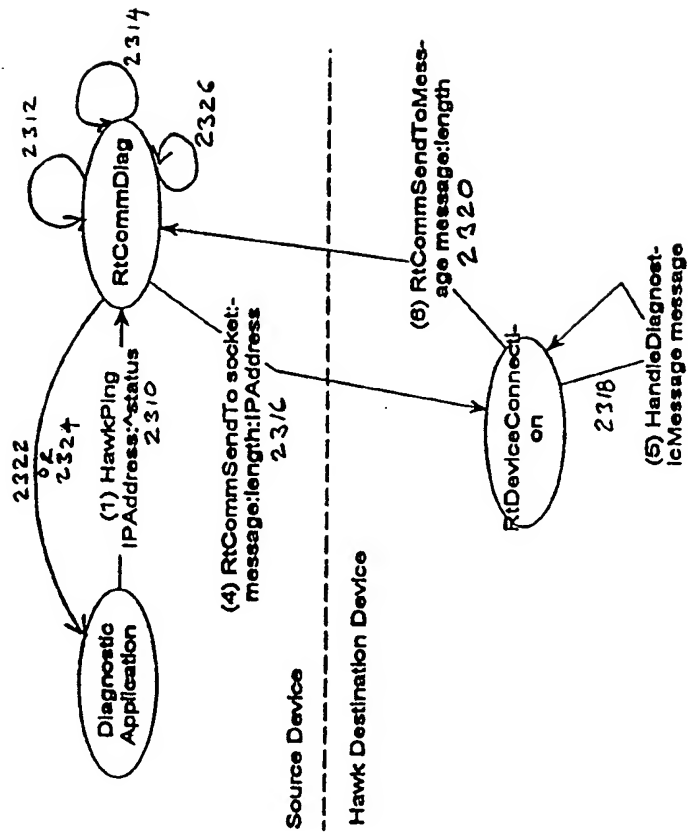


FIG. 24

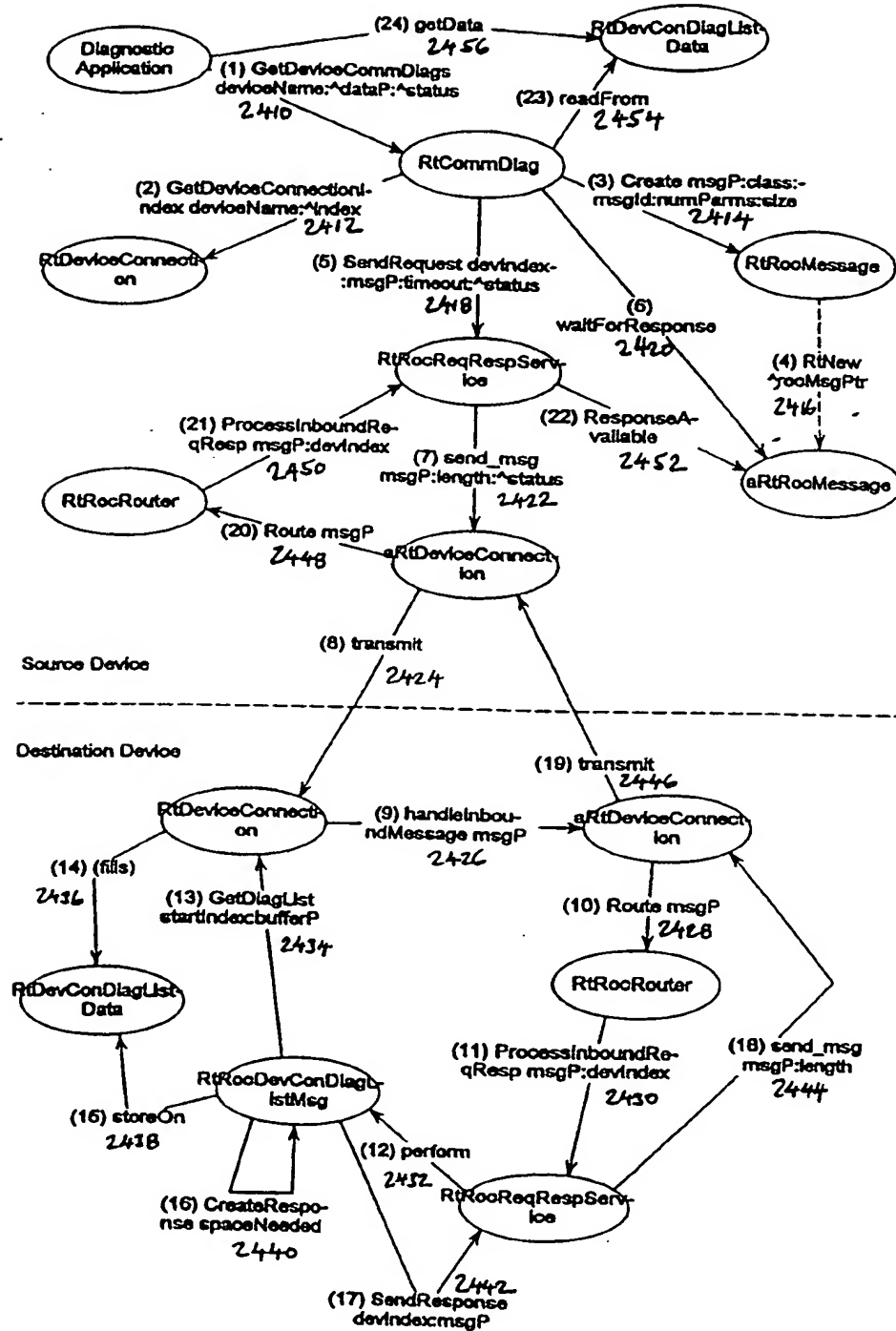


FIG. 25

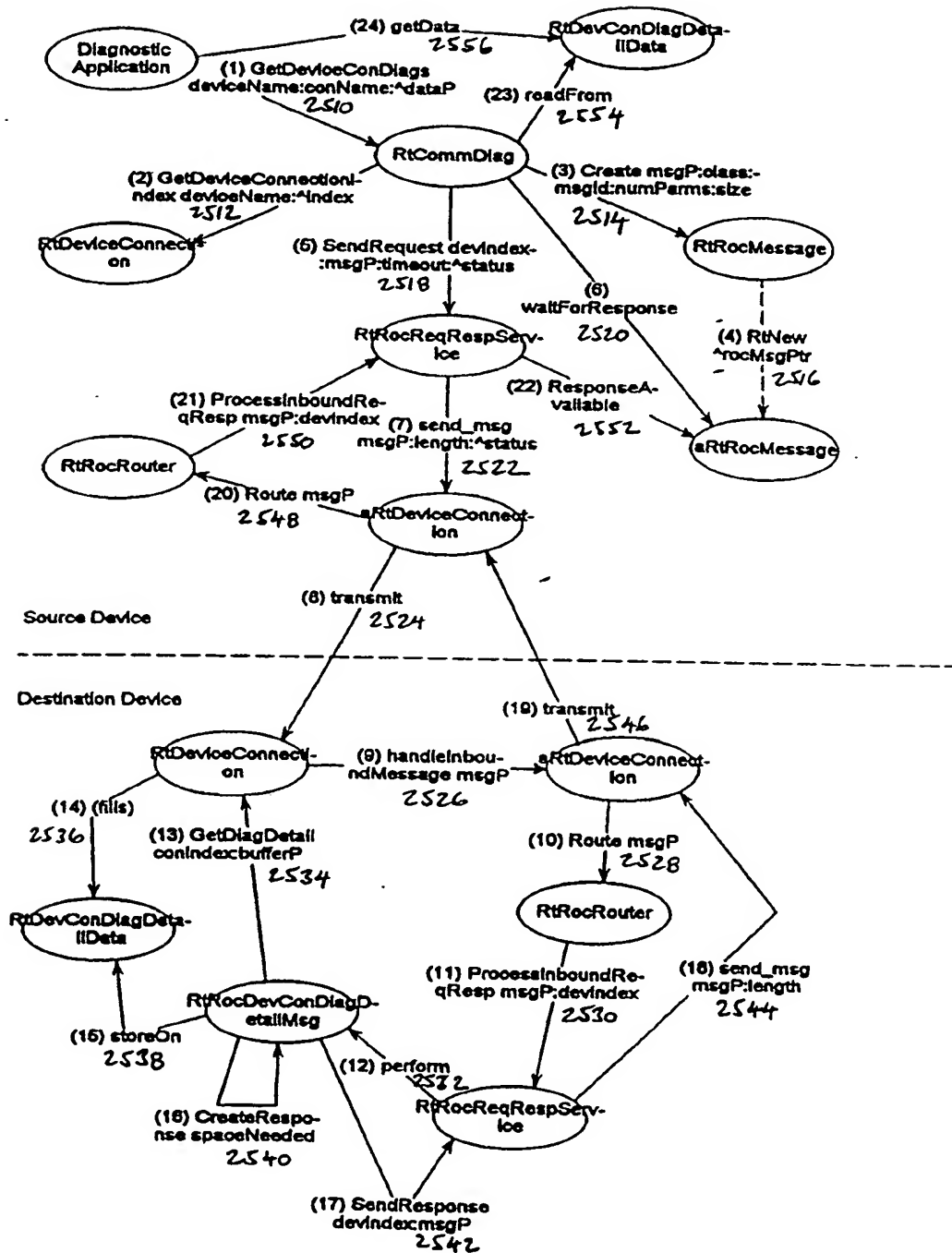
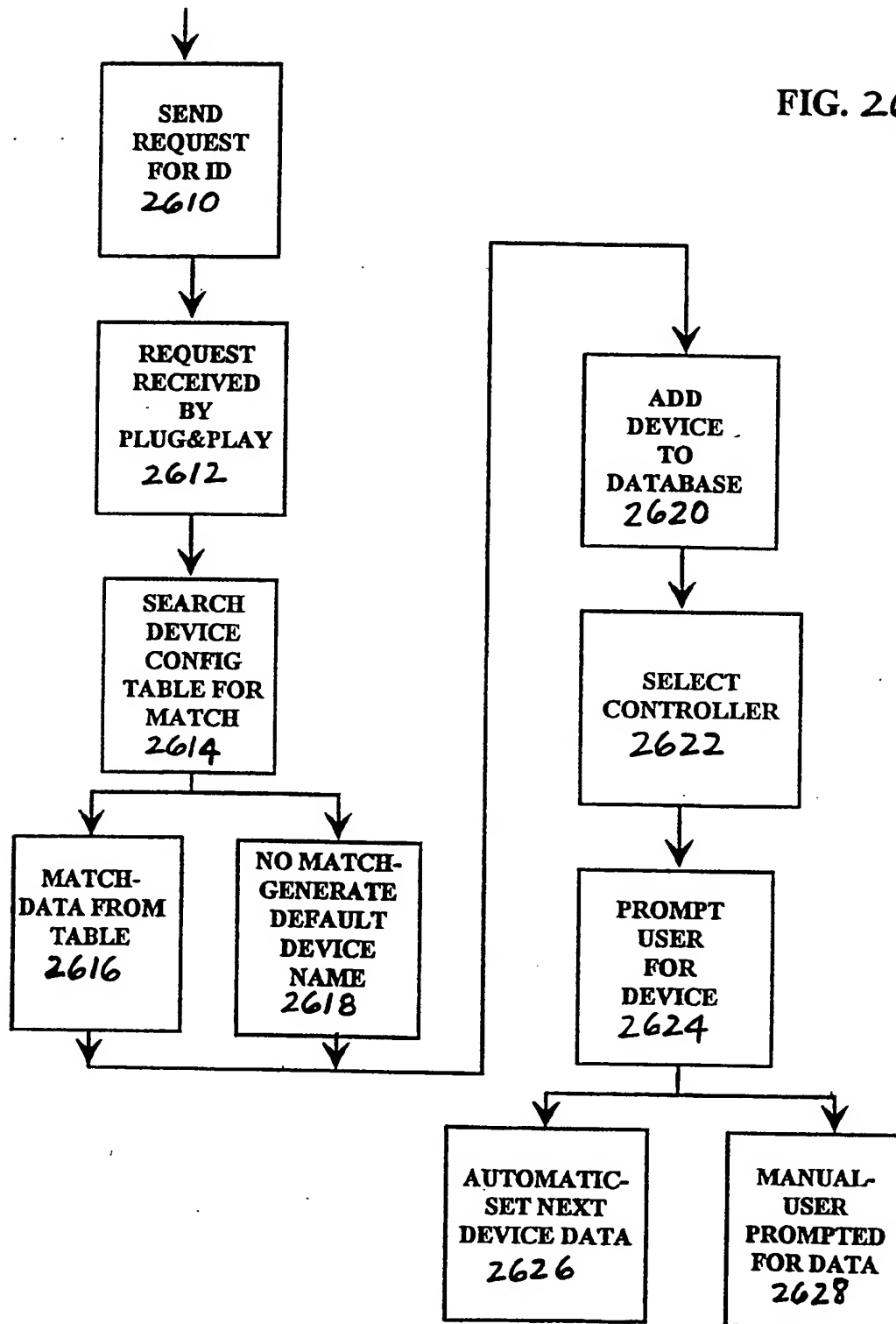


FIG. 26



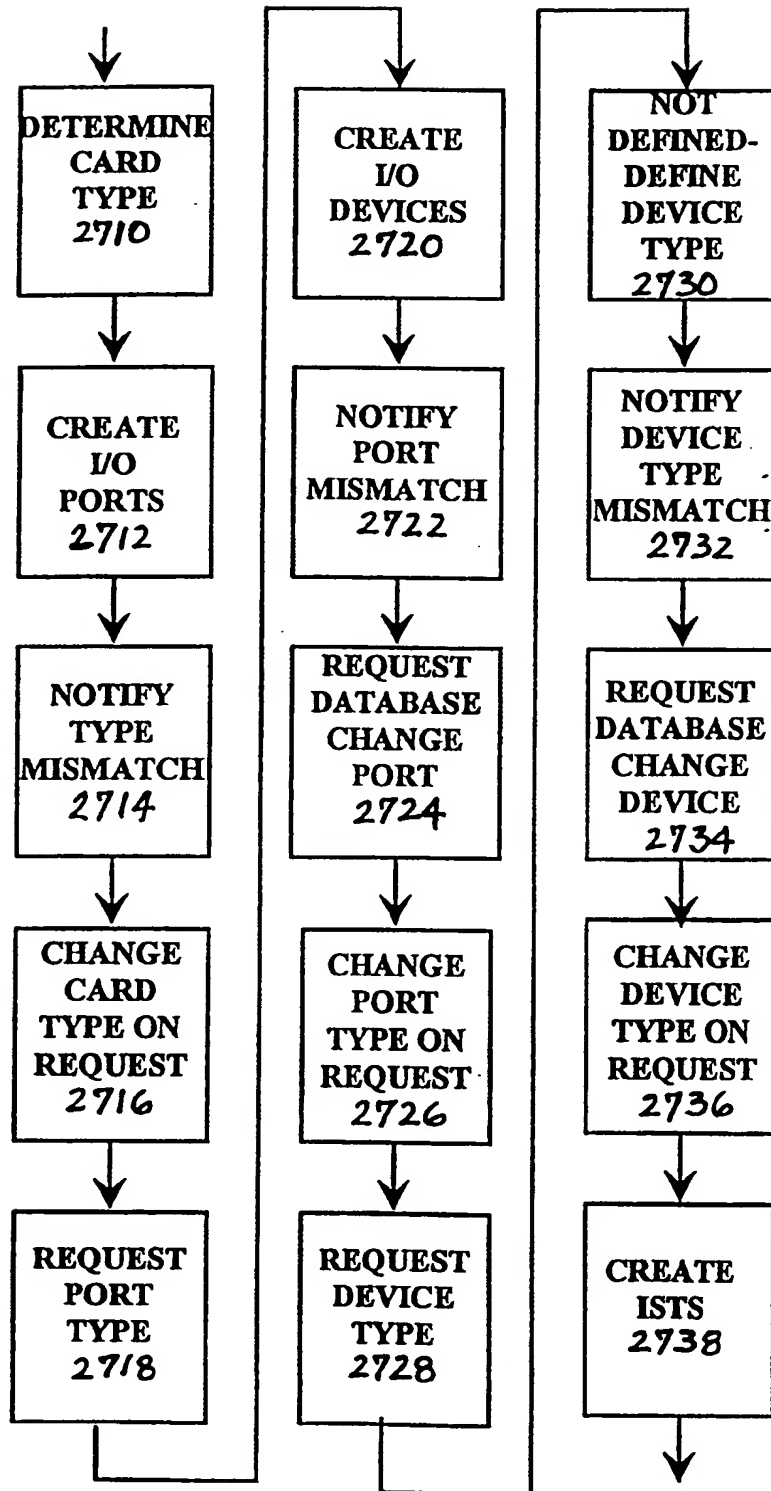


FIG. 27

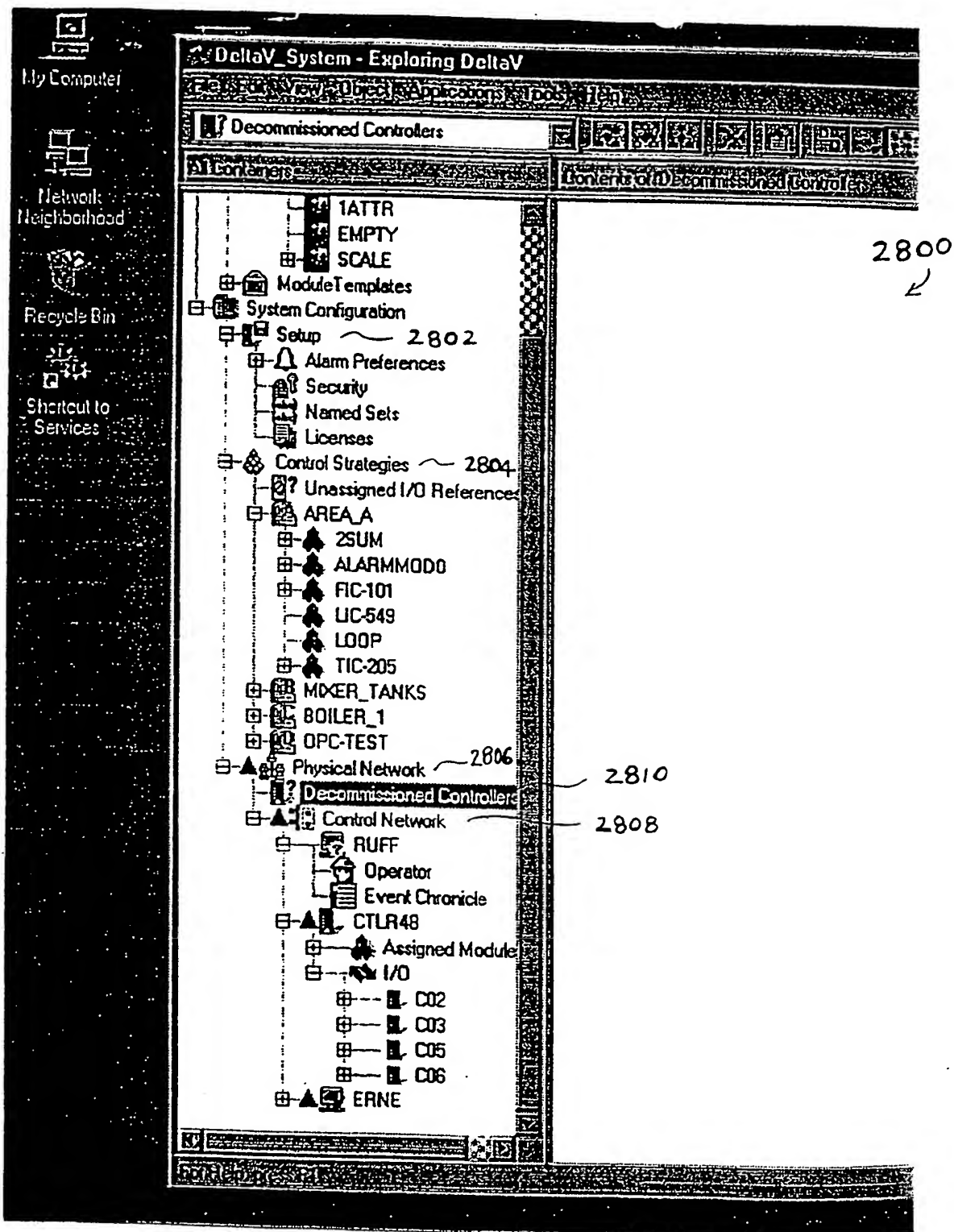


Fig. 28

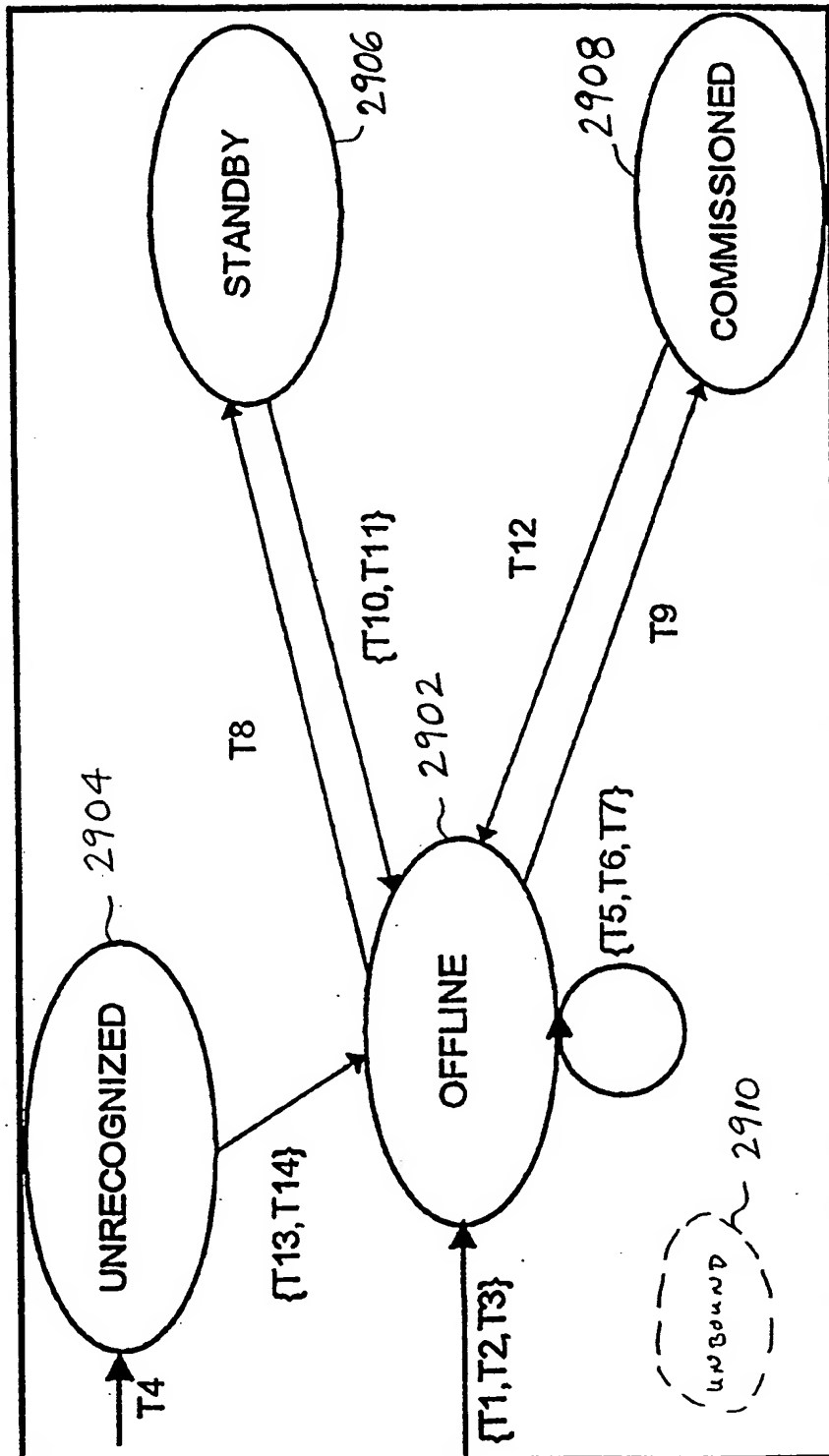


Fig. 29

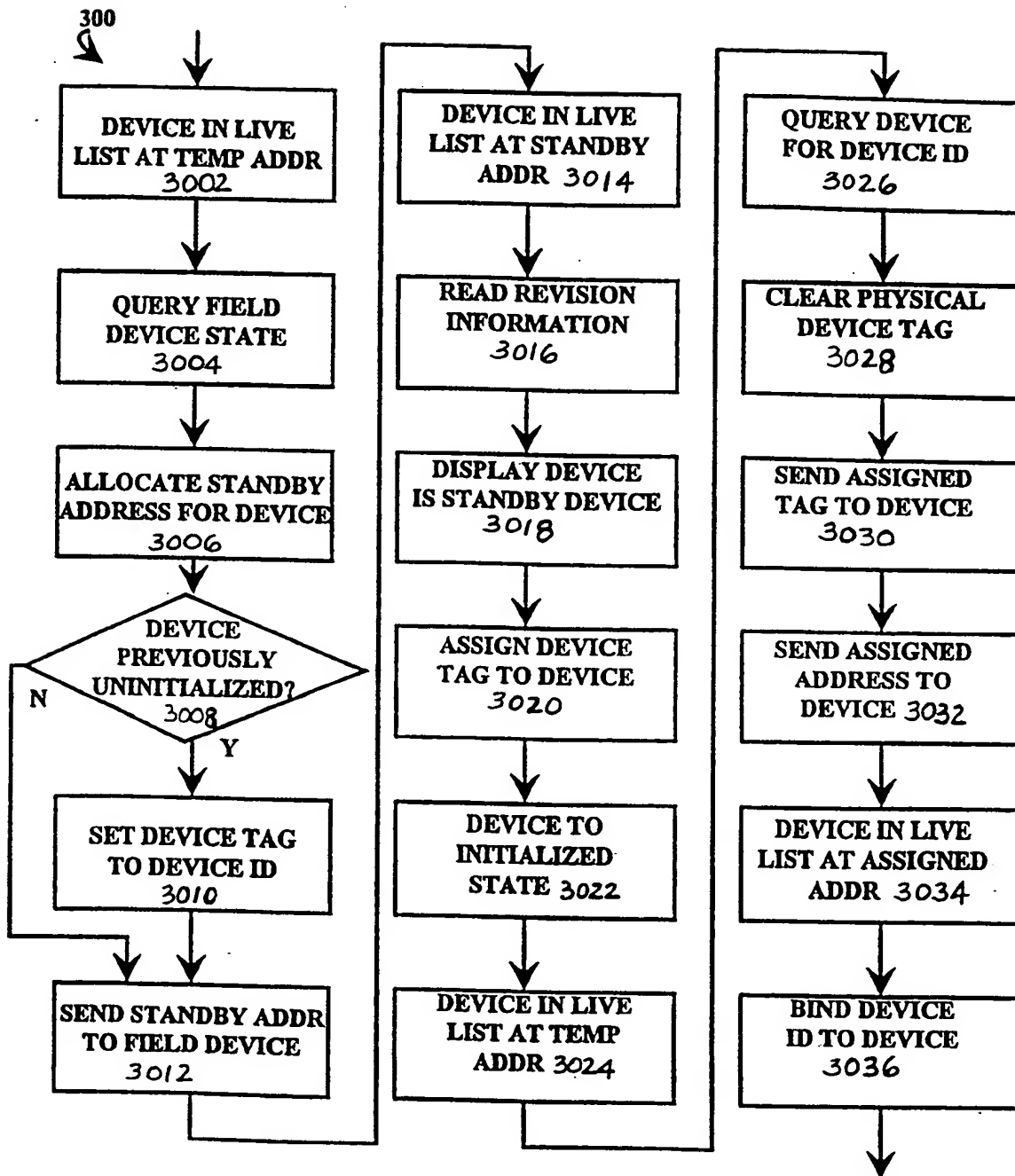


FIG. 30

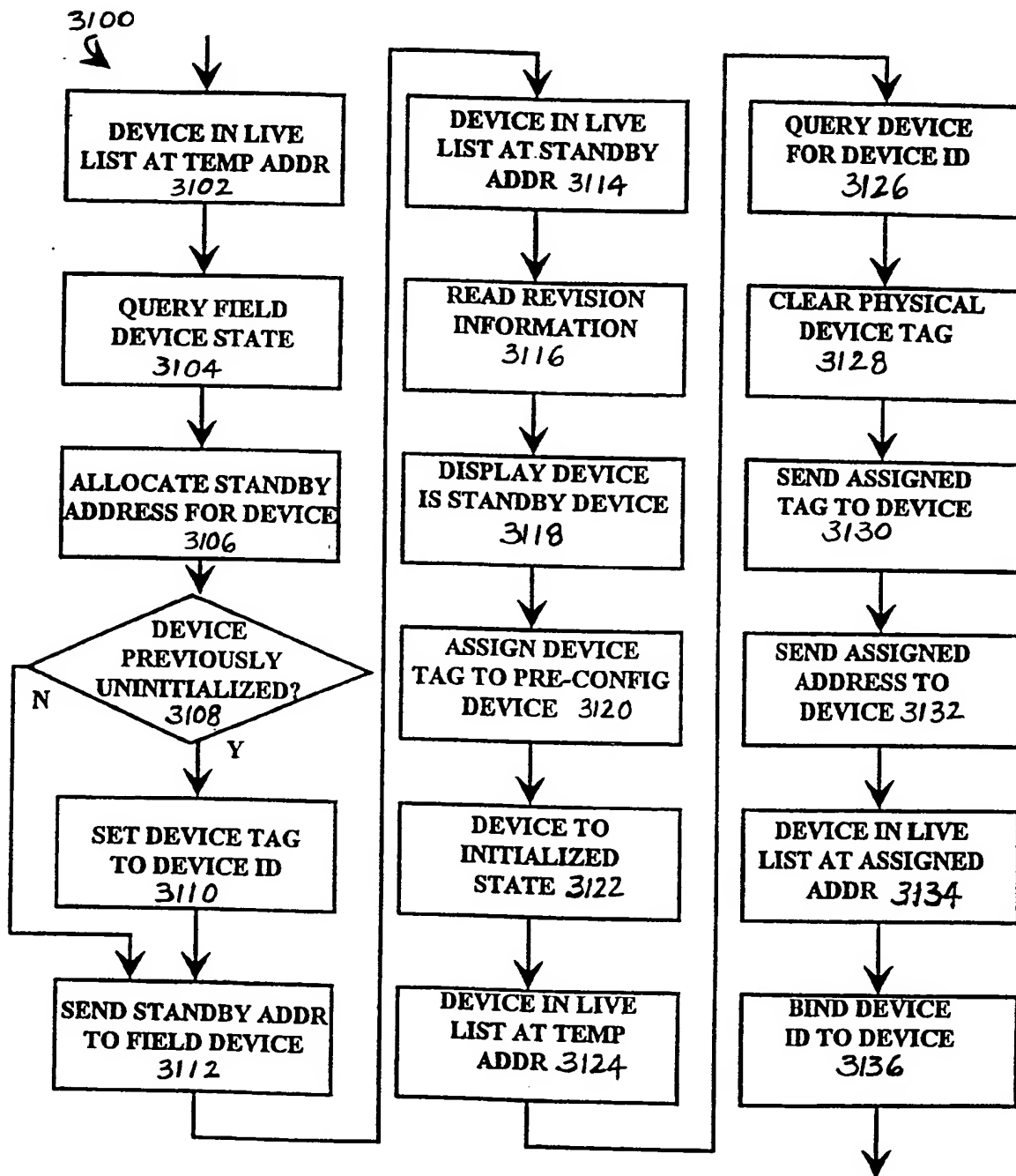


Fig. 31

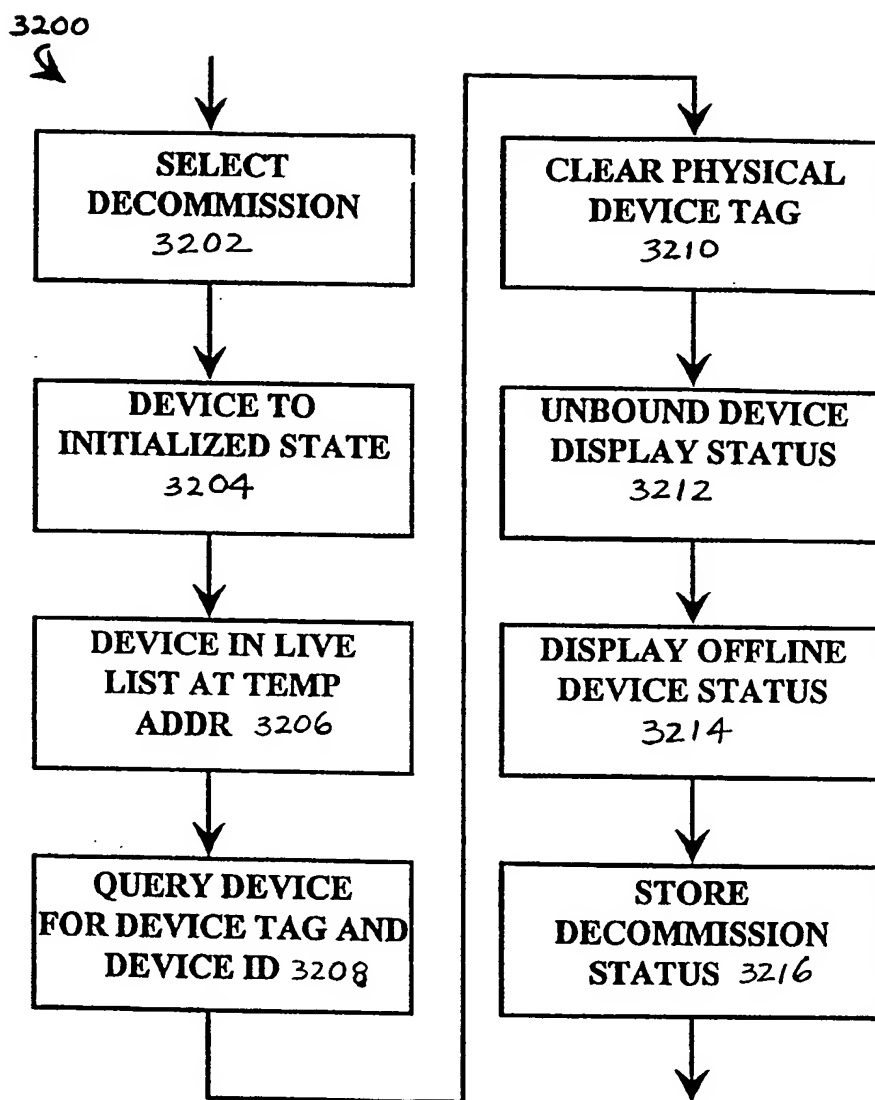


Fig. 32

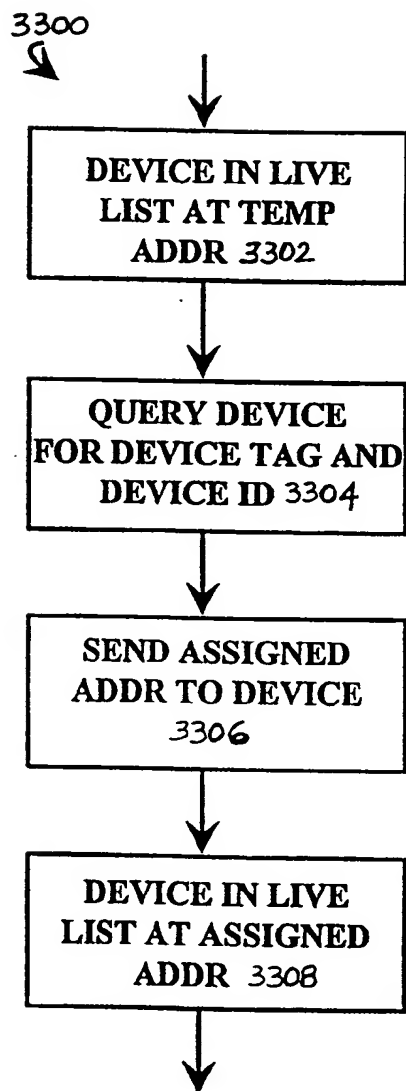


FIG. 33

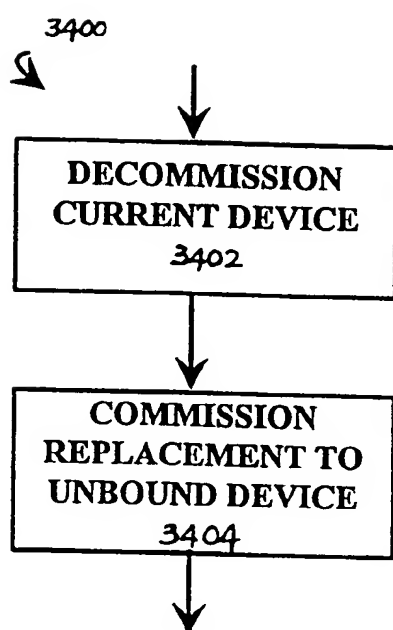


FIG. 34

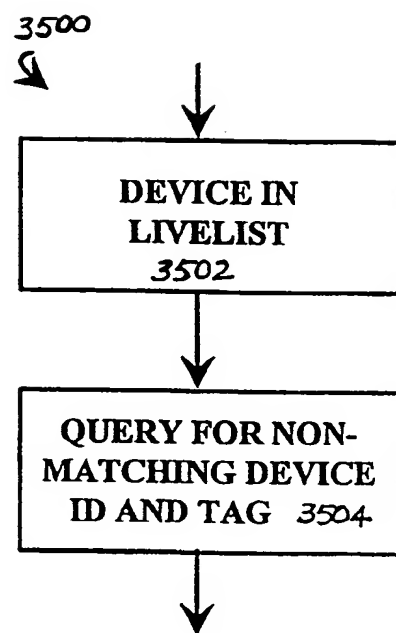


FIG. 35

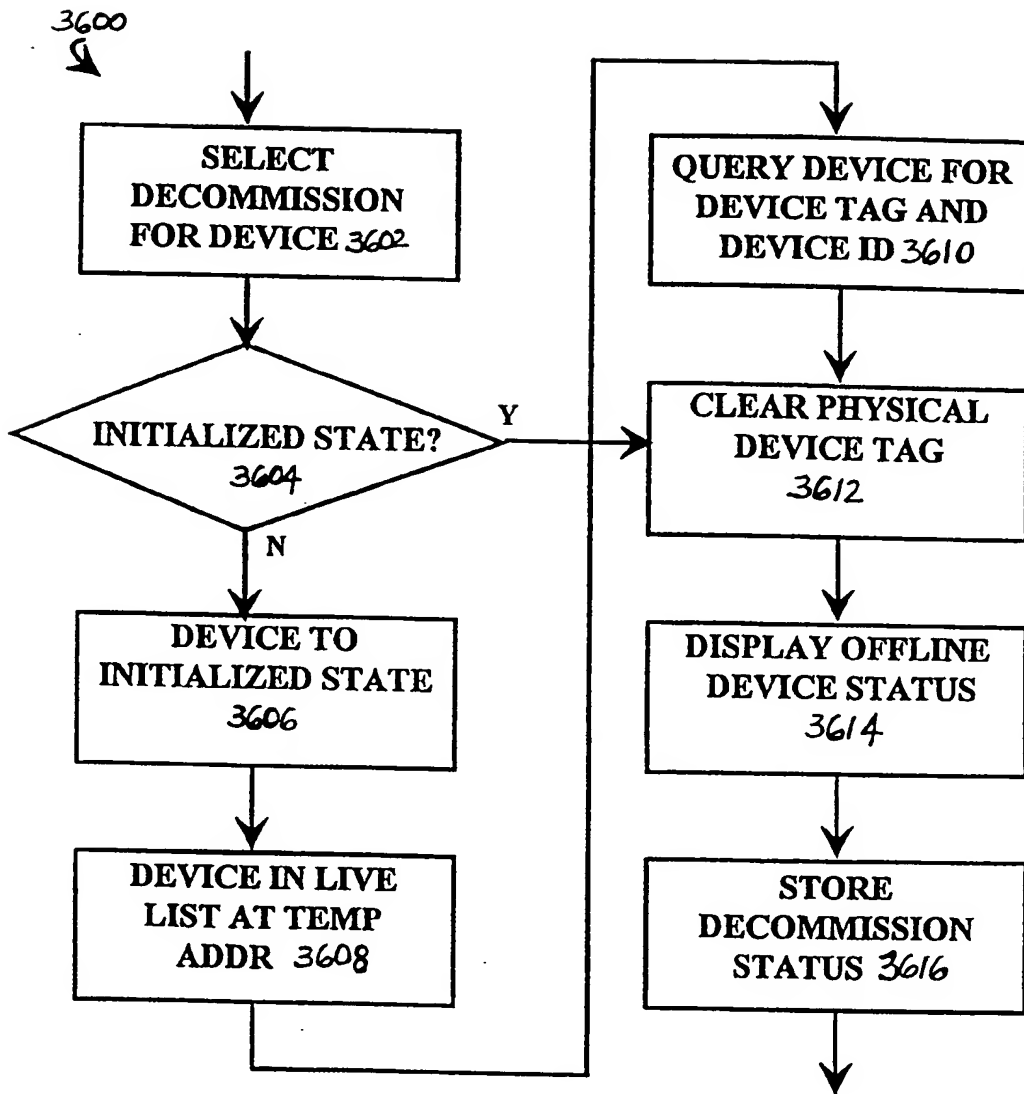


FIG. 36

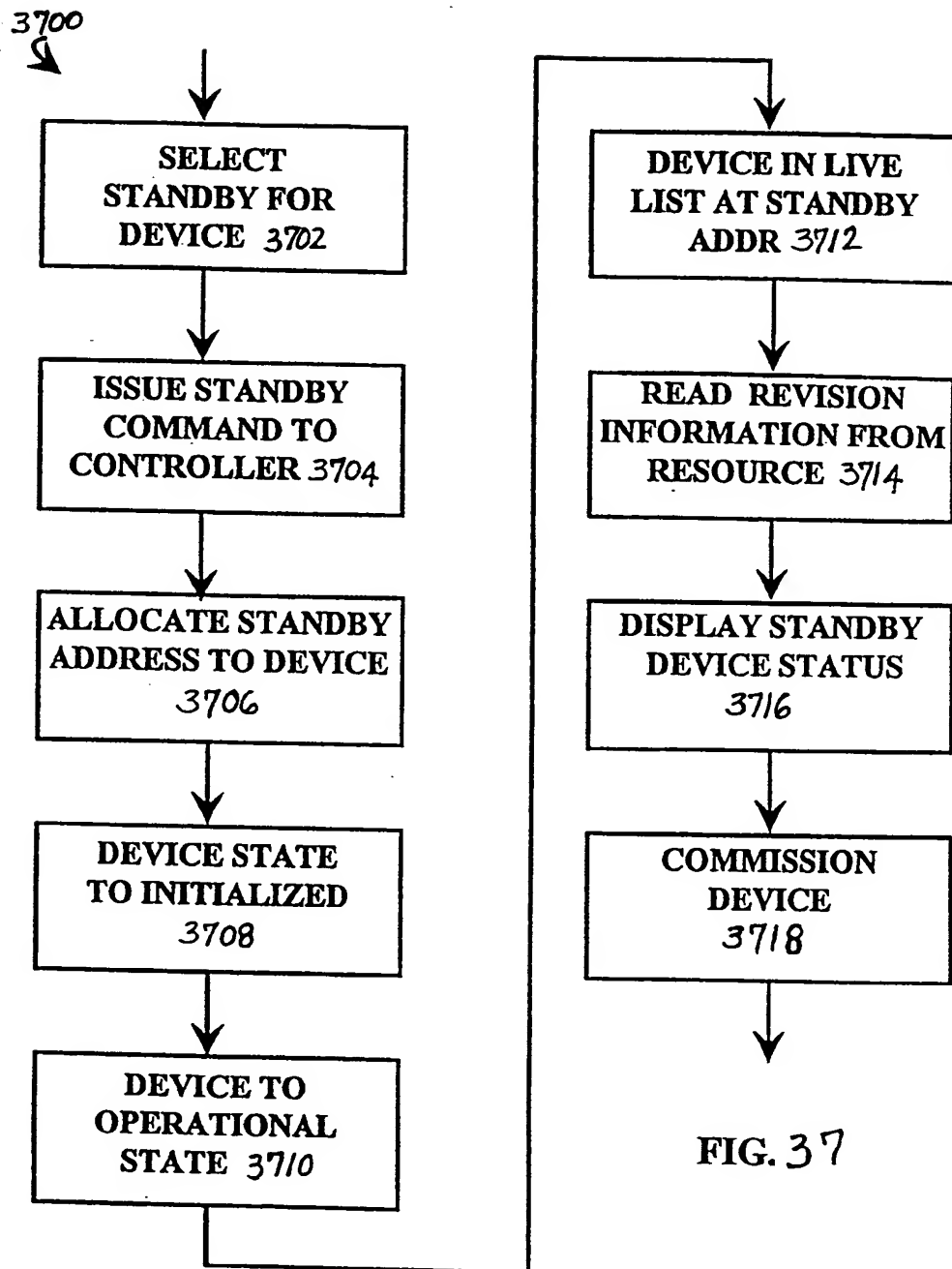


FIG. 37

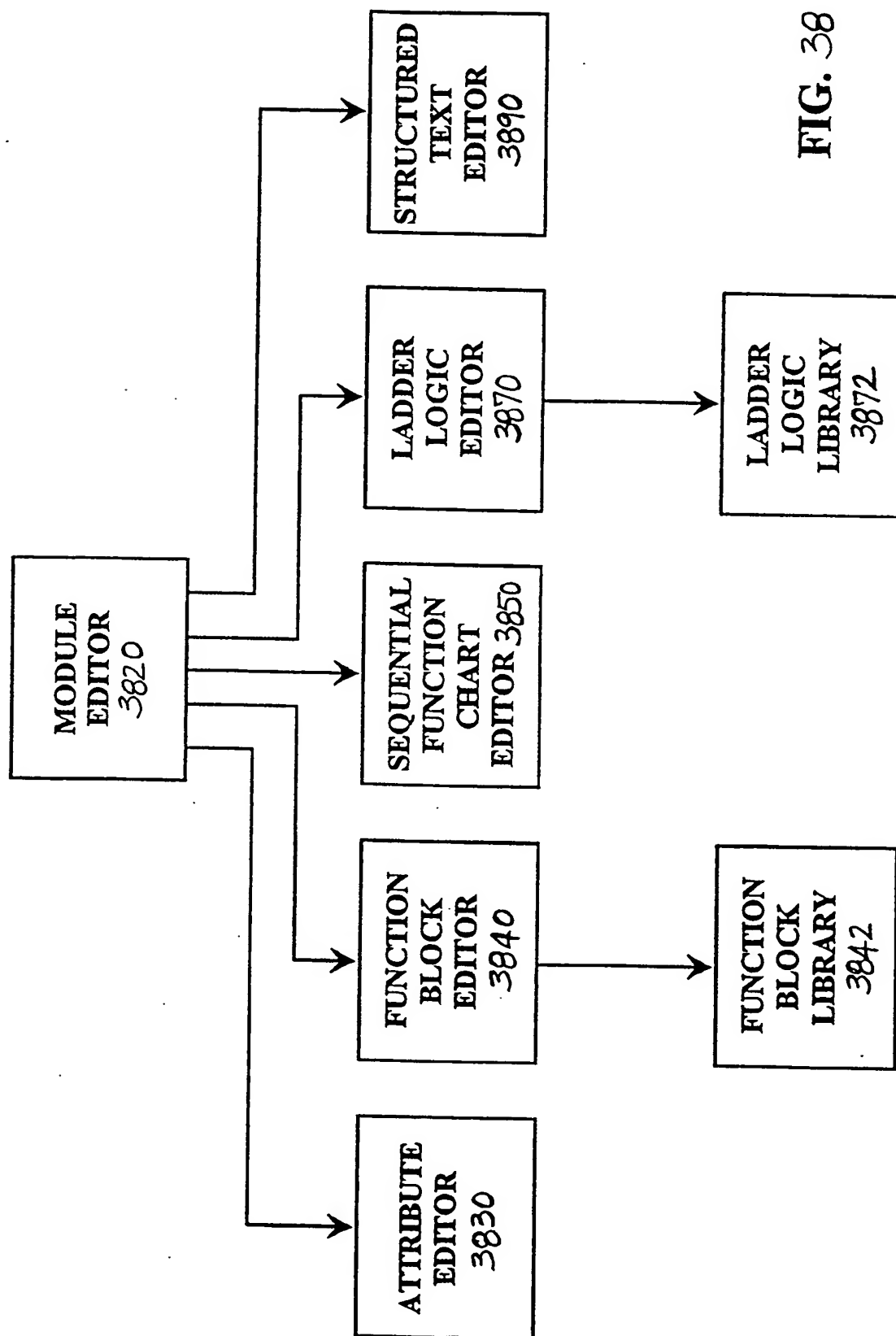


FIG. 38

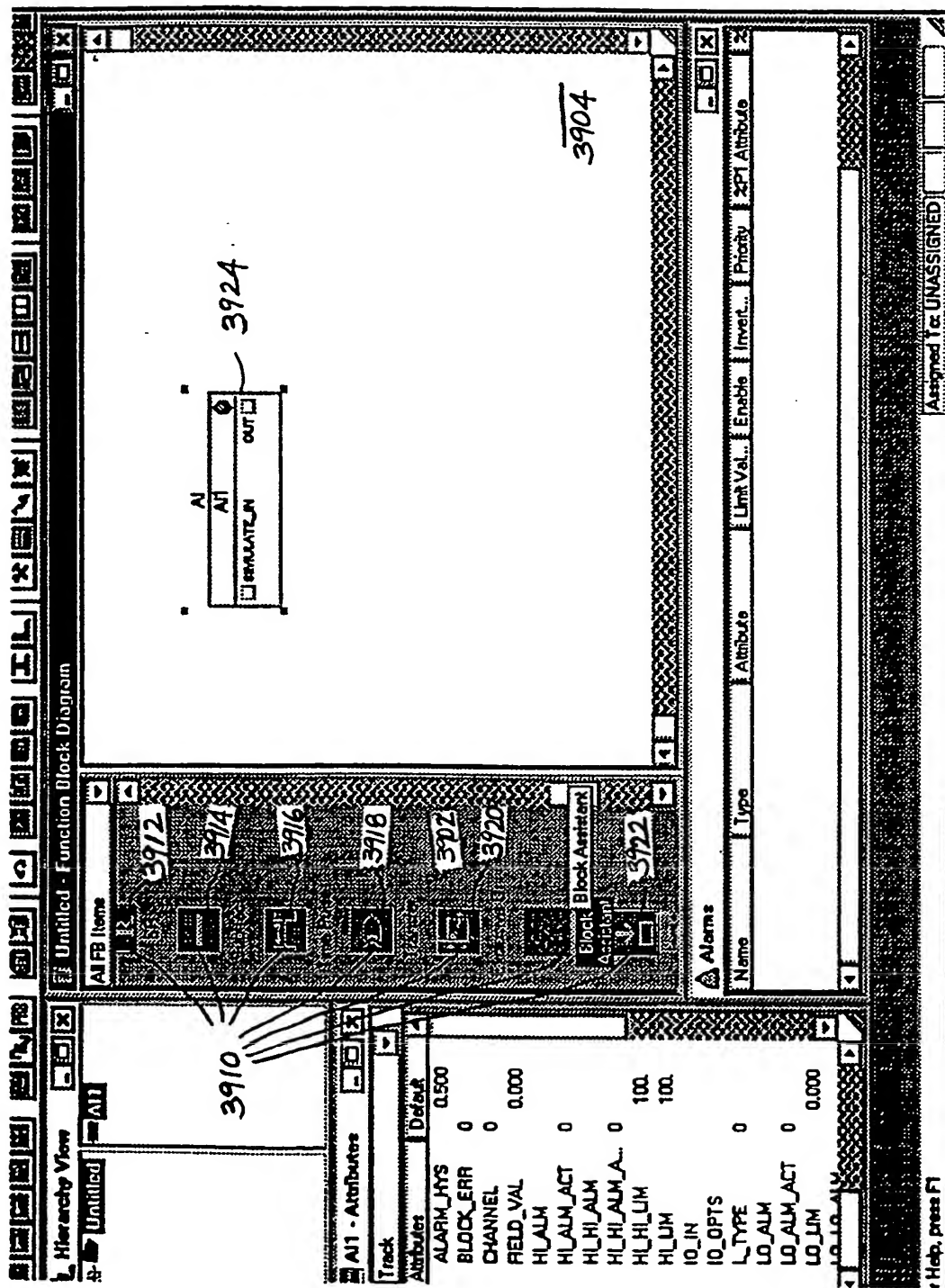


FIG. 39A

Enter the block name and select the type of block to add.

Block name:

BLOCK1

Block type:

Function Block

Embedded Block

Linked Composite

Module Block

3940

3942

3944

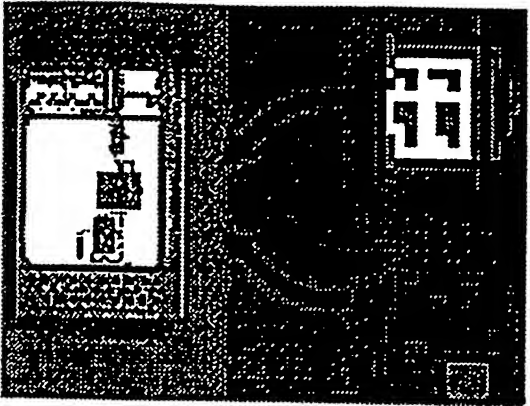
3946

< Back

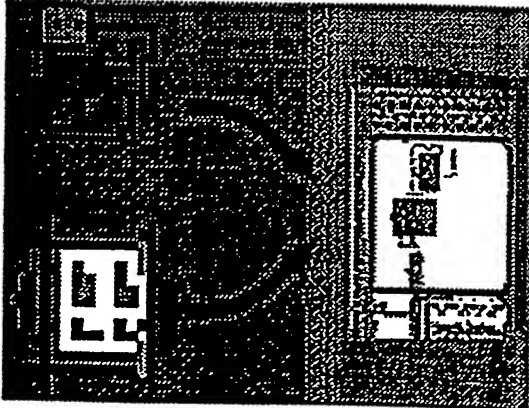
Next >

Cancel

FIG. 39B



3930
2



3960 ↙

Select the algorithm and number of inputs and outputs.

Algorithm

☒
3962 Function Block Diagram

☐
3964 Sequential Function Chart

Inputs/Outputs

inputs

outputs

< Back

Finish

Cancel

FIG. 39C

3970

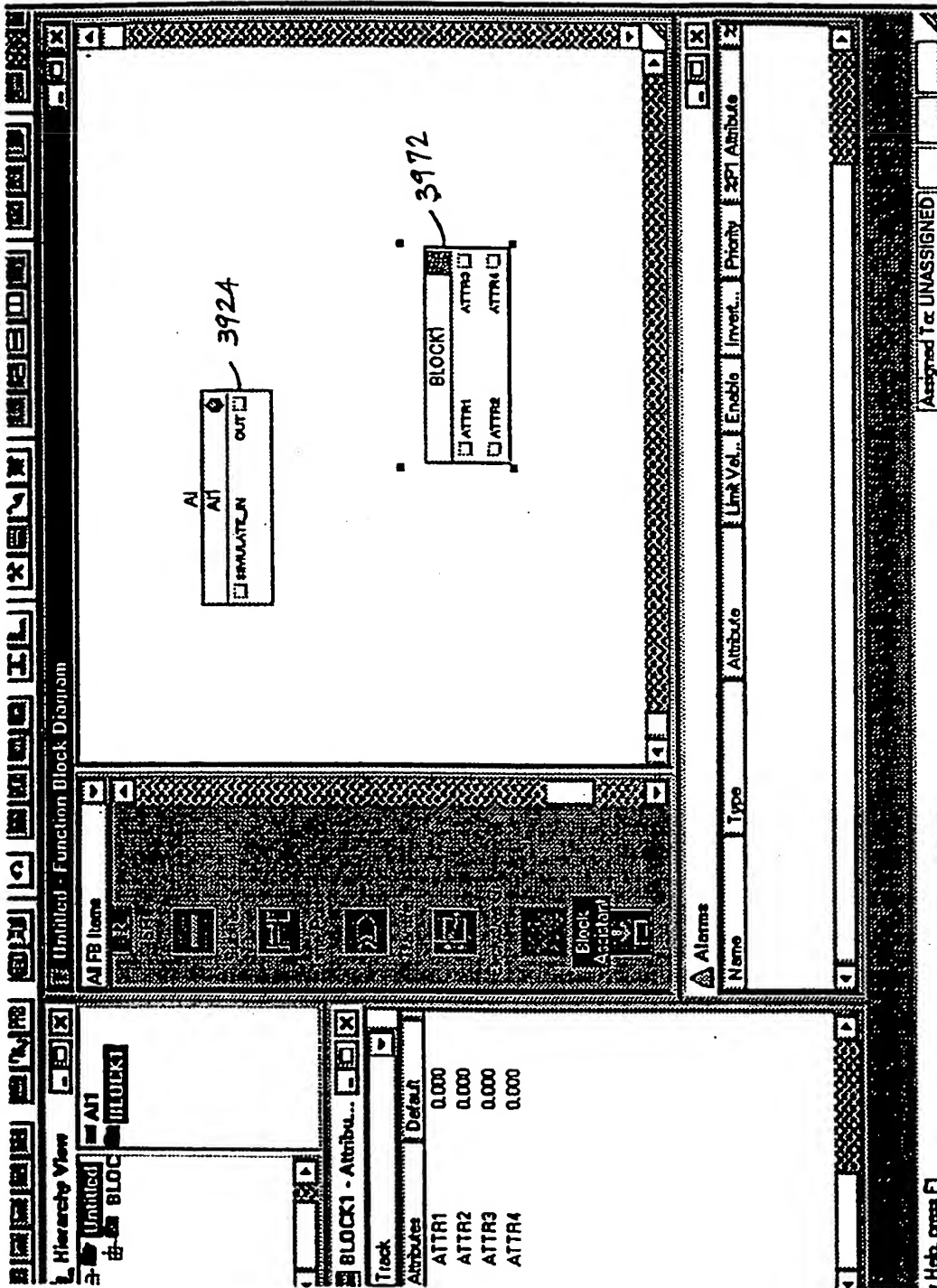


FIG. 39D

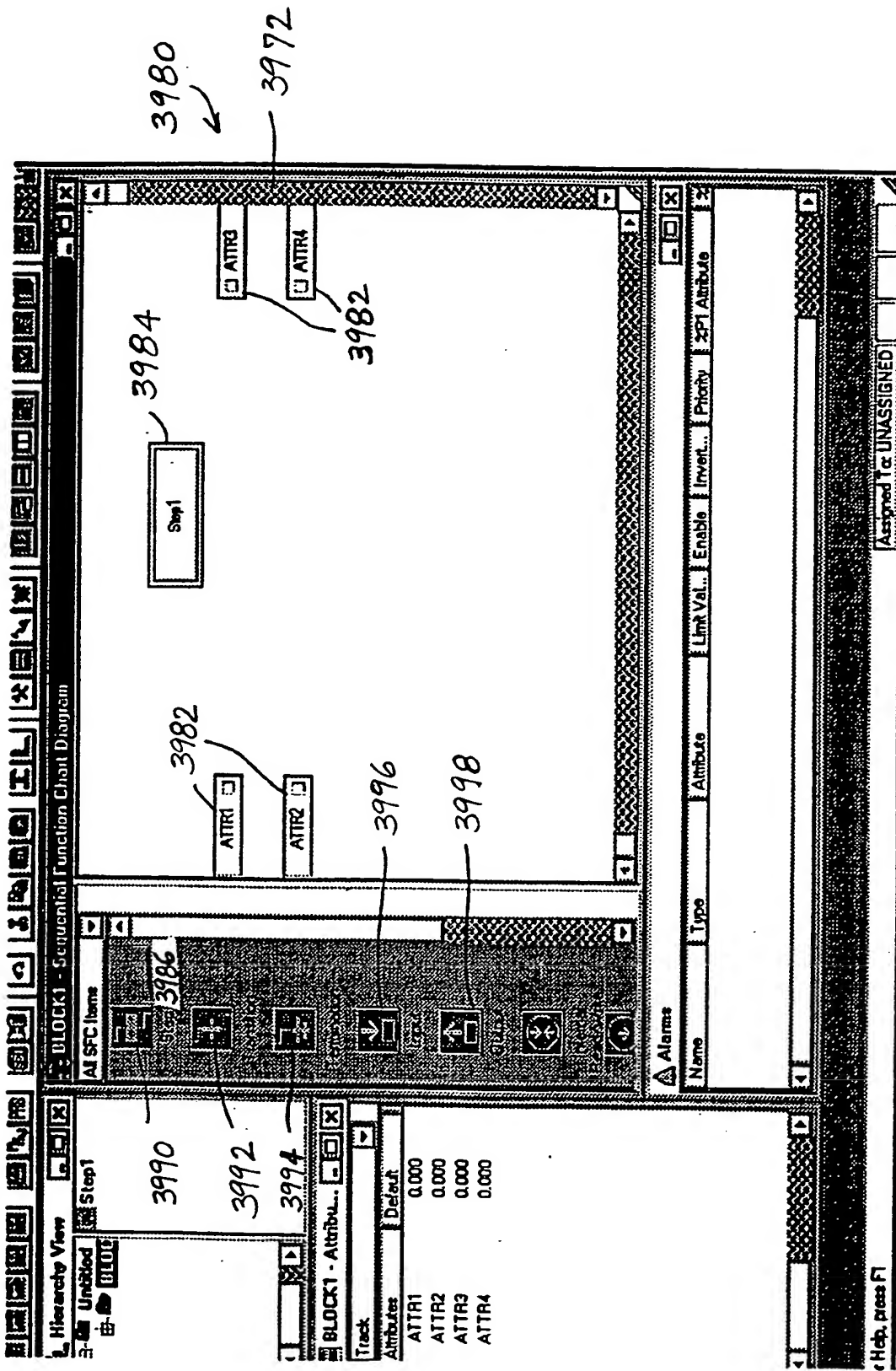
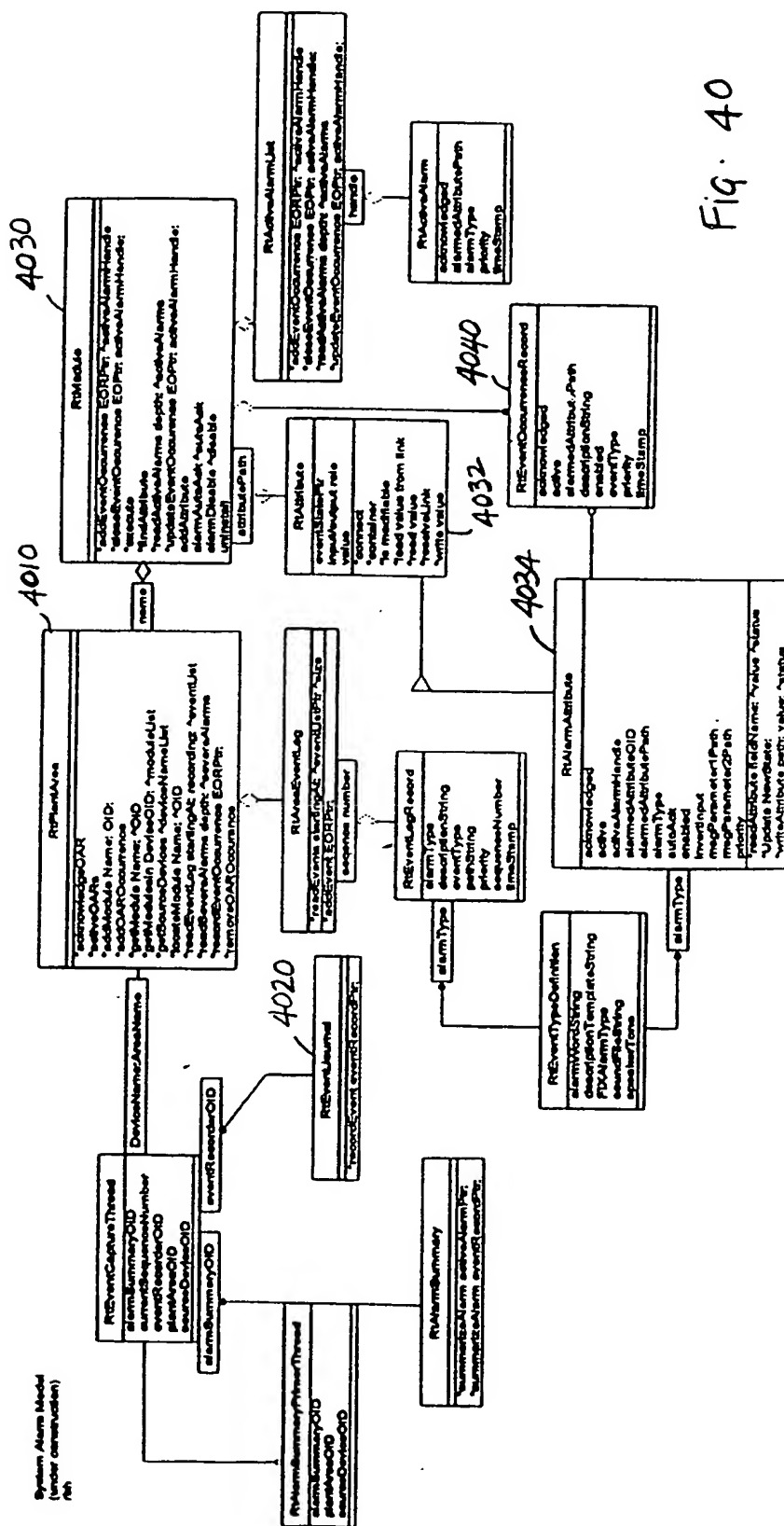
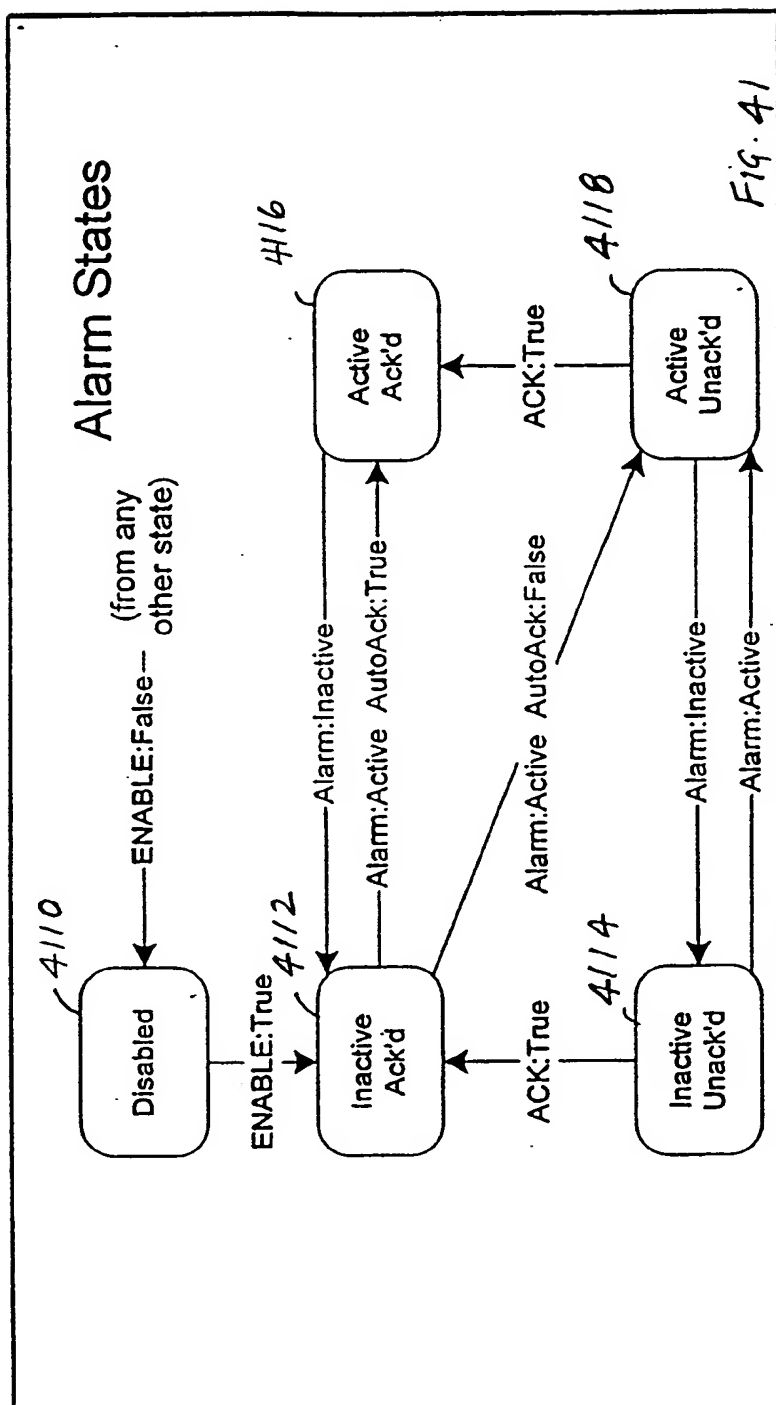


FIG. 39E





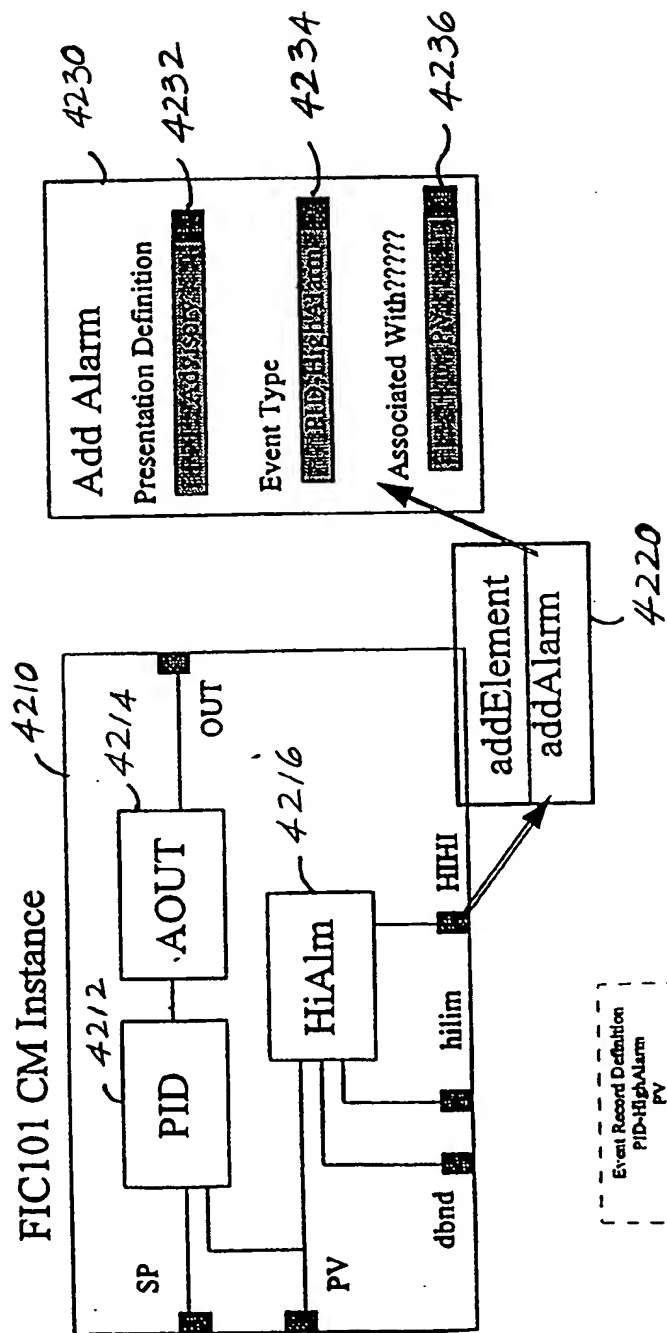


Fig. 42

Fig. 43

